TRABAJO FIN DE GRADO:

MARKERS DETECTION/RECOGNITION FOR UAVS VISUAL NAVIGATION SYSTEMS

Autor:

Raquel Mazarío Benavides

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Tutor:

Abdulla Hussein Abdulrahman Al-Kaff

Departamento de ingeniería de sistemas y automática

Universidad Carlos III de Madrid

AGRADECIMIENTOS

A mis padres Carlos y Loli, por estar a mi lado en todas mis vivencias, animarme a perseguir nuevos retos y transmitirme la idea de que con esfuerzo y trabajo soy capaz de conseguir todo lo que me proponga. También por ser los primeros en alegrarse por mis éxitos y en darme fuerzas y energía en los malos momentos.

A mi hermano Carlos por ser mi ejemplo de constancia, esfuerzo y superación. También por ser el modelo de persona que todos los días trabajo por alcanzar.

A mi abuela Pilar por desvivirse siempre por mí, por apoyarme incondicionalmente y por la complicidad y amor que nos tenemos. También por ser la que más ganas tenía de que consiguiera terminar éste trabajo.

A mis abuelos Braulio y Moni por el continuo interés que han mostrado en mi etapa universitaria, y por el amor y orgullo con que siempre hablan de mí.

A mi abuelo Rafael, con el que me hubiera encantado compartir todo el proceso que me ha llevado hasta aquí, porque sé que me ha ayudado en momentos difíciles y que me cuida desde donde esté.

A mi tía Beatriz, por interesarse por mi trabajo, por los detalles que tiene conmigo y por el cariño que siempre me demuestra.

A mi tío Jorge por preocuparse por mí y por mi futuro.

Al resto de mi familia por interesarse por mis estudios y transmitirme su apoyo.

A mis amigos, por el ánimo que me han dado, los momentos que hemos vivido, las risas y por intentar obligarme a desconectar (aunque no os haya hecho caso). Soy afortunada de teneros.

A mi tutor Abdulla, por el tiempo invertido en mí, y por sus consejos y orientación durante el desarrollo de este trabajo.

ABSTRACT

In this project, an algorithm for two-dimensional arrow detection and recognition and its subsequent direction feature extraction is presented. The extracted information will be transferred to an UAV. Using that information the UAV will be capable of following a track composed of arrows in an autonomous way, in real time, and as quickly as possible. The proposed algorithm is going to be used in autonomous drones racings and competitions.

The proposed solution is able to detect, recognize and extract information of different types of arrows. Furthermore it is scale and rotation invariant, it means that the algorithm is able to detect arrows of any size and orientation.

The algorithm can detect and recognize objects by analyzing its shape. As a result, it is not necessary to use data bases or train classifiers, so we can obtain a faster processing.

This project presents the following structure: the motivation, and objectives that has led to develop this project will be extended in the introduction. Then different existing methods used to detect and recognize objects will be explained and compared to justify why it was necessary to develop the proposed algorithm. After that, an introduction to the application will be held in order to give a detailed explanation of how the algorithm works. Experimental results, its analysis, and the budget that shows the cost of developing this project will be also included. Finally obtained conclusions will be detailed, and future works will be exposed in order to improve and extend the performances obtained with the proposed solution.

RESUMEN

En este trabajo se presenta un algoritmo para la detección y reconocimiento de flechas bidimensionales y la posterior extracción de su dirección y sentido mediante visión por computador. La información extraída se transmite a un UAV con el fin de que siga el camino marcado por las mismas de forma autónoma, a tiempo real y en el menor tiempo posible. Se busca emplear el algoritmo propuesto en carreras y competiciones de drones autónomos.

La solución propuesta es capaz de detectar, reconocer y extraer información de diferentes tipos de flecha, siendo además invariante a la escala y rotación, es decir, detecta flechas de cualquier tamaño en todas las orientaciones posibles.

El algoritmo está basado en la detección y reconocimiento del objeto mediante el análisis de su forma, gracias a lo cual no es necesario utilizar ninguna base de datos o entrenar clasificadores, lo que supone un procesamiento mucho más rápido.

La estructura de este trabajo es la siguiente: la motivación, objetivos y problemática que han llevado al desarrollo de este trabajo serán ampliados en la introducción. A continuación se expondrán diferentes métodos ya existentes que se emplean para la detección y reconocimiento de objetos, se comparará cada uno de ellos y se justificará por qué ha sido necesario implementar el algoritmo propuesto. Después se hará una introducción a la aplicación para pasar posteriormente explicar de forma detallada el funcionamiento del algoritmo. Se incluirán también los resultados experimentales y el análisis que se hace de los mismos, además de un presupuesto que recoge los gastos derivados de la realización del proyecto. Finalmente se expondrán las conclusiones obtenidas tras la realización del trabajo y se propondrán trabajos futuros con el fin de mejorar y ampliar las prestaciones que se han conseguido alcanzar con la solución propuesta.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

- UAV: Unmanned Aerial Vehicle
- UAS: Unmanned Aerial System
- AV: Array de Vértices
- VL: Vector de Lados
- VT: Vector Triángulo
- VR: Vector Rectángulo
- VD: Vértice de Dirección
- Vff: Vértices final de flecha
- Pvd: Posición de vértice de dirección
- VT1: Vector Triángulo 1
- VT2: Vector Triángulo 2
- ZT: Zona Triángulo
- ZR: Zona Rectángulo
- Pff: Posición de los Vértices final de flecha
- PM: Punto Medio
- LRY: Línea de Referencia de Yaw
- VB: Vector de Booleanas
- VA: Vector de Ángulos
- VVD: Vector de Vértices de Dirección
- FR: Flecha Reciente
- Iv: Índice de Vector

Capítulo 1: Introducción

1.1 Motivación

Los vehículos autónomos son aquellos capaces de realizar todas las tareas necesarias para su desplazamiento de un punto a otro sin intervención humana. Conseguir eliminar el papel que desempeñan las personas en las tareas de conducción o pilotaje supone importantes beneficios. En el caso de vehículos aéreos aparte de reducir los accidentes, se podría eliminar el peligro que supone enviar aeronaves pilotadas a misiones peligrosas como podrían ser las militares, las de rescate y salvamento en el caso de desastres naturales, o científicas como el seguimiento y monitorización del clima y tormentas.

Los micro UAVs, también conocidos como drones, están experimentando un creciente auge en los últimos años debido a que gracias a su pequeño tamaño y reducido coste pueden llevar a cabo tareas que otros tipos de UAVs no son capaces de implementar, o que lo harían de forma notablemente más costosa. Un estudio realizado por Juniper Research [10] estima que se vendieron cuatro millones de unidades de drones alrededor del mundo en el año 2015. No sólo eso sino que debido al gran abanico de posibilidades que se contemplan si se consigue que los drones sean capaces de navegar eficazmente de forma autónoma, estudios como el realizado por la firma Tractica [25] estiman un crecimiento exponencial en los ingresos generados por las ventas de drones que alcanzará los cuatro billones de dólares anuales en 2025.

Sin embargo la capacidad de navegación autónoma es actualmente un reto aún difícil de alcanzar. Una manera que se tiene de impulsar las tecnologías necesarias para lograrla es la de organizar competiciones. En los últimos años se han creado una gran cantidad de ellas alrededor del mundo enfocadas a desarrollar las tecnologías que supongan la mejora del vuelo autónomo de los micro UAVs, y a fomentar el desarrollo de los sistemas de visión por computador que les permitan captar información de su entorno, analizarla y actuar conforme a ella. Con ello se espera que en un futuro próximo los drones sean capaces de realizar de forma completamente autónoma tareas como el envío de pequeños paquetes a domicilios, idea que estudia la gran empresa Amazon, vigilar lugares púbicos en el ámbito de la seguridad, o una de las más complejas, utilizarlos en tareas de rescate y salvamento de personas.

Algunas ejemplos de competiciones de drones autónomos son: Sparkfun Competition, organizada por la empresa Sparkfun Electronics, IROS promocionada por el Ministerio de Comercio, Industria y Energía de Korea, UAV Challenge, organizada por empresas norteamericanas y centros de investigación, IARC “International Aerial Robotics Competition” creada en el campus del instituto de tecnología de la universidad de Georgia o IMAV “International Micro Air Vehicle Conference and Competition”.

En todas estas competiciones hay desafíos y pruebas de distinta índole, tanto para drones pilotados como autónomos. Una de las competiciones para drones autónomos más destacadas es IROS “Autonomous Drone Racing Challenge”, surgida con el objetivo de brindar a los investigadores de robótica de todo el mundo una exhibición de vuelo autónomo y promover el desarrollo de soluciones para un vuelo ágil en ambientes arriesgados. Algunos de los desafíos técnicos de los que está compuesta

esta competición son detección de obstáculos, detección de marcadores para el guiado del vuelo del dron o localización y detección de fallos y recuperación.

Este trabajo se va a orientar a desarrollar un algoritmo que permita superar el segundo desafío, en el que el dron debe ser capaz de seguir un camino indicado por marcadores (en este caso flechas) gracias a la visión por computador. Siendo el ganador el que consiga recorrerlo correctamente en menos tiempo.

Figura 1. Carrera de drones [1]

1.2 Problemática

El problema planteado es que un dron sea capaz de superar una prueba de una competición de drones autónomos que consiste en seguir un camino marcado por flechas pintadas en el suelo, siendo el ganador el que lo recorra en el menor tiempo. Las bases de esta prueba establecen que las flechas que formen el camino podrán ser de diferentes tamaños y formas, siendo todas ellas de un mismo color uniforme.

Por tanto deben emplearse técnicas de visión por computador que permitan detectar, reconocer y extraer información de diferentes tipos de flechas presentadas en cualquier orientación y de cualquier tamaño con el fin de poder efectuar el guiado del dron. Todo ello deberá ser a tiempo real y empleando el menor tiempo posible.

1.3 Objetivos

Como se verá en el *Capítulo 2*, la mayoría de los algoritmos empleados para detección y reconocimiento de objetos emplean bases de datos con numerosas plantillas del objeto que se quiere buscar, lo que supone un esfuerzo de análisis considerable que aumenta en gran medida el tiempo de procesamiento, llegando algunos de ellos incluso a no trabajar a tiempo real. En este trabajo se está desarrollando un algoritmo para carreras de drones autónomos, por lo que ahorrar tiempo en un aspecto fundamental.

Por las razones expuestas, y por las que se van a ampliar en el *Capítulo 2* el objetivo de este trabajo es desarrollar un algoritmo que sea capaz de detectar y reconocer flechas de diferentes formas, tamaños y en cualquier orientación para posteriormente extraer de ellas su dirección y sentido, información en base a la cual el dron efectuará los movimientos correspondientes. Por tanto también será necesario incluir el control que permita que estos movimientos se produzcan de forma correcta y precisa. Todo ello debe ser realizado a tiempo real y además en el menor tiempo posible, por lo que se va evitar hacer uso de bases de datos.

1.4 UAVs

UAV (Unmanned Aerial Vehicle) son las siglas que se utilizan para denominar a los vehículos aéreos no tripulados. Existen varias formas de que una aeronave pueda ejercer su plan de vuelo sin ningún tripulante: pueden estar guiadas por control remoto, seguir un plan de vuelo programado con anterioridad, o volar de forma autónoma gracias a sistemas más complejos y dinámicos.

En este ámbito de los vehículos aéreos no tripulados también se habla de las siglas UAS (Unmanned Aerial System) que se utilizan para denotar el sistema aéreo no tripulado, es decir el vehículo aéreo, más el sistema de control que utiliza.

Este tipo de vehículos han sido desarrollados con la intención de evitar la intervención humana en las tareas denominadas como las tres "D", en inglés *Dull, Dirty, Dangerous missions.* Es decir, en misiones aburridas, sucias o peligrosas.

Cabe destacar que se excluyen de la definición de UAV los misiles balísticos, semibalísticos, misiles crucero, proyectiles de artillería, planeadores que no llevan fuerza propulsora, globos y dirigibles que se sustentan bajo fuerzas de flotabilidad, y objetos arriostados que carecen de control remoto o autónomo.

1.4.1 Tipos

Dentro de los UAVs actualmente existen diferentes tipos *ver Tabla 1*, clasificados por peso, altitud máxima de vuelo, duración de autonomía de vuelo etc.

Tabla 1. Tipos de UAVs [6]

	Mass (kg)	Range (km)	Flight alt. (m)	Endurance (h)
Micro	<5	<10	250	1
Mini	<20/25/30/150[a]	<10	150/250/300	<2
	Tactical			
Close range (CR)	25–150	10–30	3000	2–4
Short range (SR)	50–250	30–70	3000	3–6
Medium range (MR)	150–500	70–200	5000	6–10
MR endurance (MRE)	500–1500	>500	8000	10–18
Low altitude deep penetration (LADP)	250-2500	>250	50–9000	0.5–1
Low altitude long endurance (LALE)	15-25	>500	3000	>24
Medium altitude long endurance (MALE)	1000-1500	>500	3000	24–48
	Strategic			
High altitude long endurance (HALE)	2500-5000	>2000	20000	24-48
Stratospheric (Strato)	>2500	>2000	>20000	>48
Exo-stratospheric (EXO)	TBD	TBD	>30500	TBD
	Special task			
Unmanned combat AV (UCAV)	>1000	1500	12000	2
Lethal (LET)	TBD	300	4000	3–4
Decoys (DEC)	150-250	0–500	50-5000	<4

[a] Varies with national legal restrictions

Dentro de la clasificación de micro/mini UAVs se encuentran los drones.

1.4.2 Drones

Los drones son vehículos aéreos no tripulados de tamaño y peso reducido, entorno al medio metro de longitud y rondando los 50 gramos. Están dotados de una serie de sensores que le permiten estabilizar y controlar su vuelo, llevando incorporada la mayoría de modelos una cámara de vídeo que les permite captar información de su entorno. La duración de la batería que utilizan como fuente de energía está entorno a los 15 – 30 minutos. Debido a su reducido tamaño y a los elementos de los que está formado su precio es considerablemente más barato que el resto de tipos de UAVs, rondando el intervalo de los 200 - 1000 euros. Cabe destacar que todas estas cifras son estándar y variarán dependiendo de la marca y el modelo del dron elegido.

Puede haber drones de alas fijas similares a las de un avión *ver Figura 2*, o drones multirotor siendo los más comunes los de cuatro hélices llamados cuadricópteros, aunque también existen los de seis (hexacópteros) *ver Figura 3.*

Figura 2. Dron de alas fijas [20]

Figura 3. (a) Cuadricóptero [22] y (b) Hexacóptero [8]

Los drones de competición difieren un poco de los anteriores, en cuanto a que deben presentar características más especiales. Las fundamentales son: velocidad, resistencia y facilidad/agilidad a la hora de sustituir piezas ya sea por avería, mantenimiento o mejoras que se quieran implementar.

El chasis de este tipo de drones debe ser preferiblemente de fibra de carbono por su ligereza y resistencia. Además son de un tamaño mucho más reducidos que los drones “de calle” para ganar en ligereza y velocidad *ver Figura 4.*

Figura 4. Modelo de drones de competición [26]

1.4.2.1 Aplicaciones

En este apartado se van a desarrollar algunas de las numerosas aplicaciones [12] actuales y futuras que pueden llevar a cabo los drones gracias a las cámaras y diferentes sensores que llevan instalados:

- Militar:
  - Defensa
  - Seguridad y control fronterizo
- Civil:
  - Seguridad: Vigilancia de lugares públicos. Monitorización de grandes multitudes (conciertos, manifestaciones…)
  - Desastres naturales y ayuda humanitaria: Ayudan en las tareas de búsqueda, rescate y salvamento de personas.
  - Movilidad y tráfico: Monitorización y gestión de la situación del tráfico.
- Ciencia:
  - Medio ambiente: Seguimiento de la meteorología, tormentas, control del estado de la atmósfera…
  - Cartografía: Entre otras cosas, sirven para la realización de mapas y modelos de elevación del terreno.
  - Servicios forestales: Control de incendios, seguimiento de áreas boscosas etc.
  - Geología: Realización de mapas geológicos, sedimentológicos etc. Control y monitorización de explotaciones mineras y su impacto ambiental (movimientos de tierras, residuos metálicos, balsas de decantación…) Caracterización de zonas con riesgos de aludes utilizando cámaras multiespectrales para medir la humedad de la nieve, y térmicas para determinar su temperatura.
  - Zoología: Seguimiento de animales en su medio natural.
- Comercial:
  - Agricultura: Gestión de cultivos, control y monitorización del estado de los mismos, control de la eficiencia de regadío, conteo y supervisión de la producción agrícola etc.
  - Transporte: Para llevar cargas, transportar y entregar mercancías.
  - Cine y deportes extremos: Son utilizados para grabar escenas cinematográficas, o acrobacias en deportes como surf, esquí, snowboard…

1.4.2.2 Normativa

En este apartado se van a exponer algunos de los puntos para la regulación del uso de drones en España, recogidos en [3] y [13]. Cabe destacar que estos puntos sólo se aplican al uso de aeronaves pilotadas por control remoto de peso inferior a 150 kg y con fines comerciales o profesionales.

Podrán realizarse actividades aéreas de trabajos técnicos o científicos por aeronaves civiles pilotadas por control remoto, de día y en condiciones meteorológicas visuales con sujeción a los siguientes requisitos:

- *Sólo podrán operar en zonas fuera de aglomeraciones de edificios en ciudades, pueblos o lugares habitados o de reuniones de personas al aire libre, en espacio aéreo no controlado, más allá del alcance visual del piloto, dentro del alcance de la emisión por radio de la estación de control y a una altura máxima sobre el terreno no mayor de 400 pies (120 m), las aeronaves civiles pilotadas por control remoto cuya masa máxima al despegue sea inferior a 2 kg, siempre que cuenten con medios para poder conocer la posición de la aeronave.*

Asimismo, podrán realizarse los siguientes tipos de vuelos por aeronaves civiles pilotadas por control remoto, de día y en condiciones meteorológicas visuales, en espacio aéreo no controlado, dentro del alcance visual del piloto, o, en otro caso, en una zona del espacio aéreo segregada al efecto y siempre en zonas fuera de aglomeraciones de edificios en ciudades, pueblos o lugares habitados o de reuniones de personas al aire libre:

a) Vuelos de prueba de producción y de mantenimiento, realizados por fabricantes u organizaciones dedicadas al mantenimiento.
b) Vuelos de demostración no abiertos al público, dirigidos a grupos cerrados de asistentes a un determinado evento o de clientes potenciales de un fabricante u operador.
c) Vuelos para programas de investigación, nacionales o europeos, en los que se trate de demostrar la viabilidad de realizar determinada actividad con aeronaves civiles pilotadas por control remoto.
d) Vuelos de desarrollo en los que se trate de poner a punto las técnicas y procedimientos para realizar una determinada actividad con aeronaves civiles pilotadas por control remoto previos a la puesta en producción de esa actividad.
e) Vuelos de I+D realizados por fabricantes para el desarrollo de nuevos productos.
f) Vuelos de prueba necesarios para demostrar que las actividades solicitadas conforme al apartado 3 pueden realizarse con seguridad.

Para consultar la normativa completa *ver* [3] y [13].

1.5 Planificación

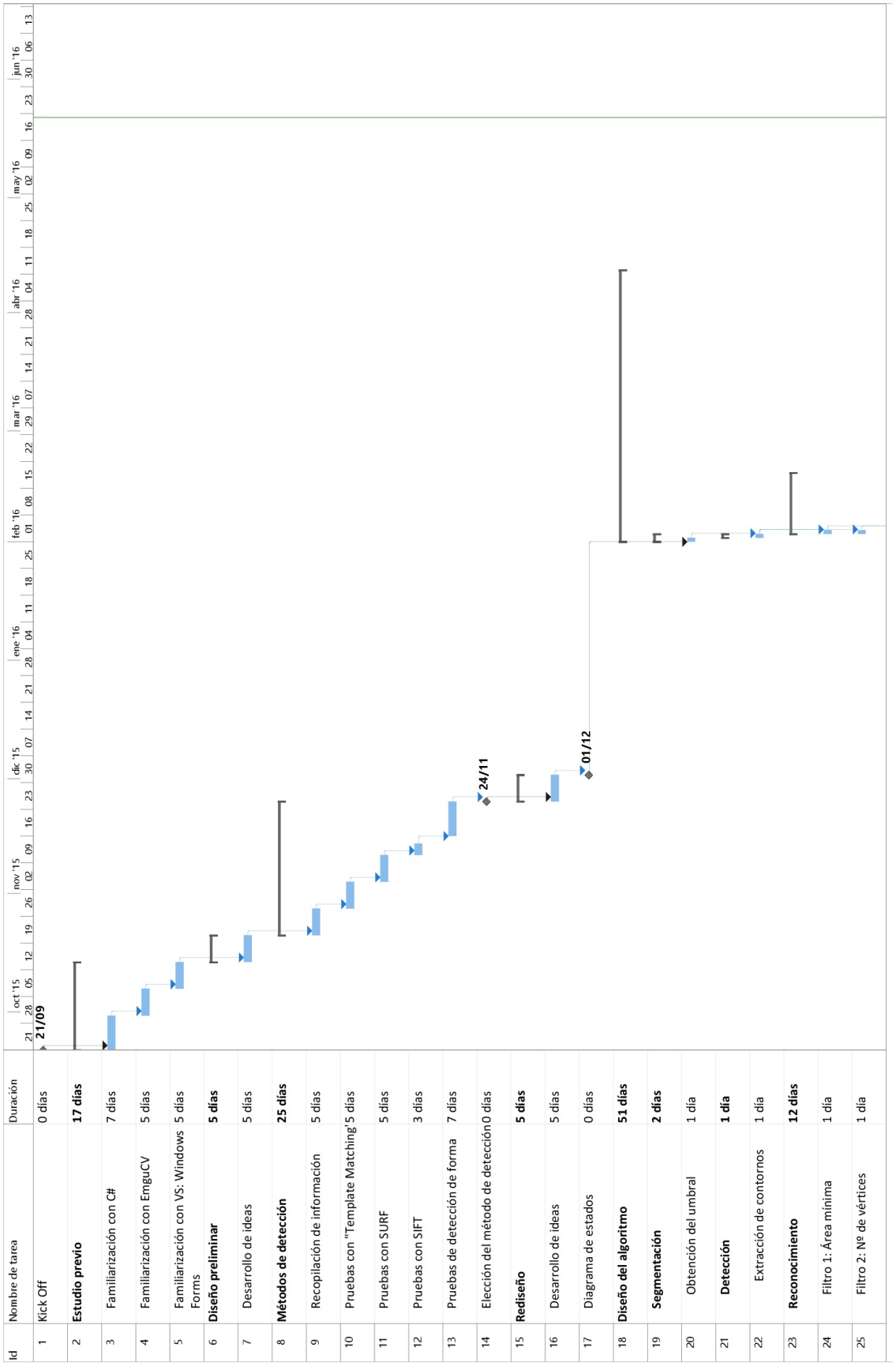


Figura 5. Planificación 1/2

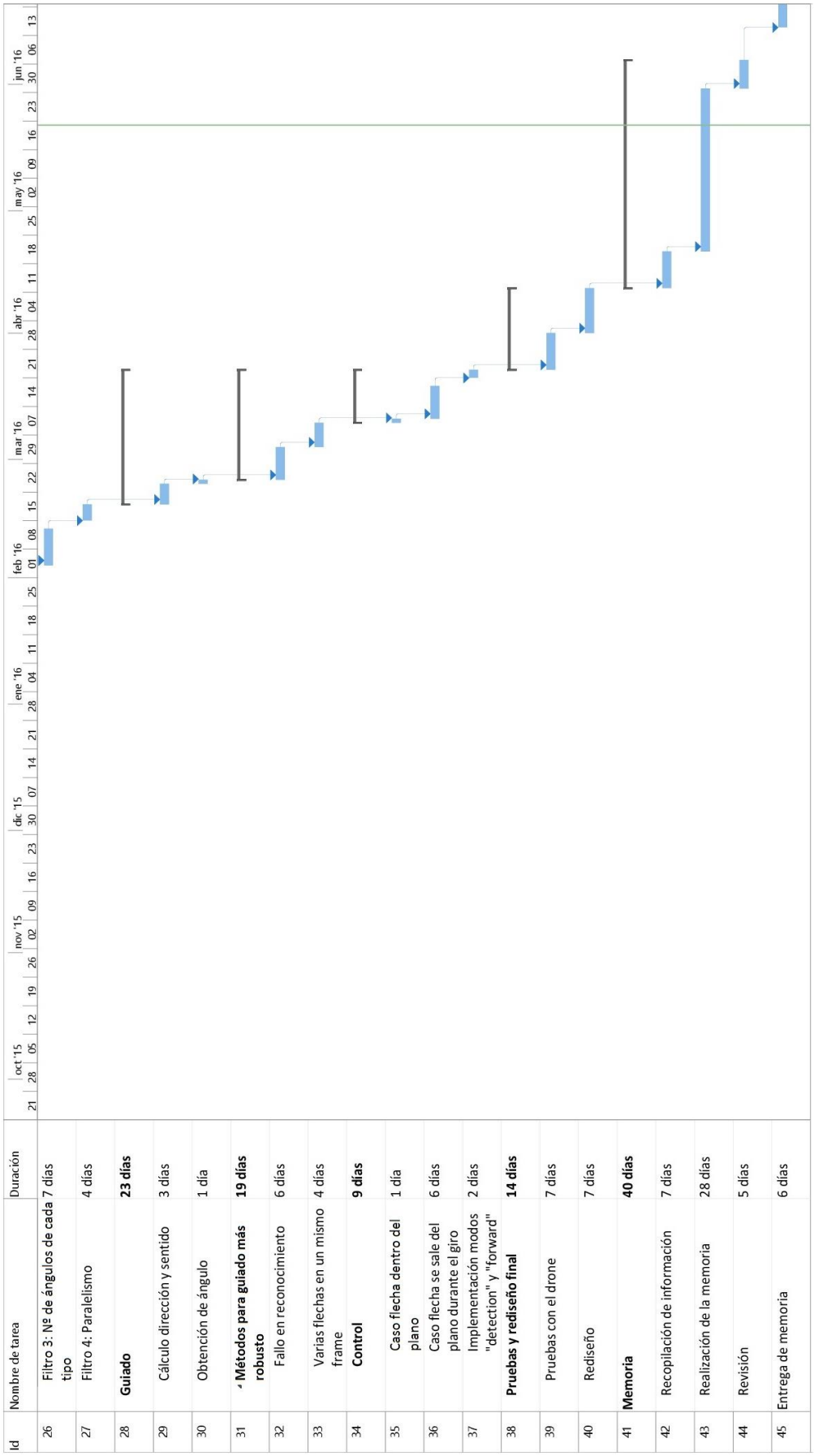


Figura 6. Planificación 2/2

Como se puede observar en las *Figuras 5 y 6*, la duración total del proyecto ha sido de ocho meses. Los dos primeros se dedicaron a la recopilación de información acerca del lenguaje de programación, las librerías de visión por computador y el entorno de desarrollo. Además se hizo el diseño preliminar de la solución y se probaron diferentes métodos de detección de objetos estudiando sus pros y contras para finalmente elegir el método que se emplearía en la solución final.

El mes de diciembre fue no laborable. En los cuatro meses posteriores se hizo un rediseño de la solución basado en el método de detección elegido, se programó el algoritmo, se hicieron pruebas con el dron, y se volvió a rediseñar y reprogramar en base a los resultados de esas pruebas.

Finalmente los últimos meses se dedicaron a la recopilación de información para el desarrollo de la memoria y a la redacción de la misma.

Capítulo 2: Estado del Arte

2.1 Métodos

En este capítulo se van a exponer algunos de los métodos que se utilizan en visión por computador para la detección de objetos y formas. Se dará una pequeña explicación de cómo funcionan para después enumerar los inconvenientes que tienen para la consecución del objetivo de este trabajo, expuesto en el *apartado 1.3*. De esta manera se justificará por qué ha sido necesario desarrollar el algoritmo propuesto como solución en este trabajo.

2.1.1 Template Matching

En este apartado se van a explicar las limitaciones que tiene detectar objetos con el método de "**`MatchTemplate()`**" incluido en las librerías de EmguCV.

Este método parte de una plantilla que contiene el objeto que se quiere buscar y la va comparando con el plano captado por la cámara para analizar las coincidencias. En la parte derecha de la *Figura 7* se muestra la plantilla con el objeto buscado, mientras que en la parte izquierda se observa que se ha producido una detección, ya que lo mostrado en la plantilla coincide con lo captado por la cámara.

Al utilizar únicamente una plantilla como modelo y no una base de datos compuesta por gran cantidad de ellas, es un método bastante rápido porque no requiere casi esfuerzo de procesamiento.

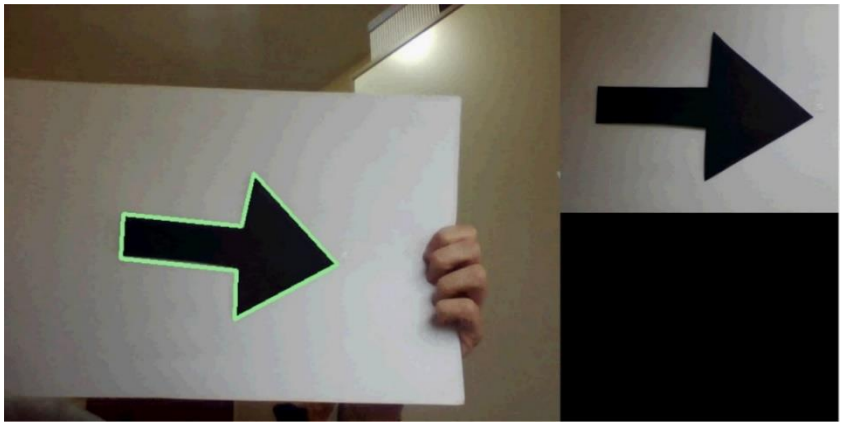


Figura 7. Match Template: Detección

Inconvenientes: Este método no es invariante ni a la rotación ni a la escala, es decir, si el objeto se encuentra rotado con respecto a la plantilla *ver Figura 8,* o en una escala mayor o inferior a la plantilla *ver Figuras 9 y 10* respectivamente, no se detecta coincidencia entre la plantilla y el plano analizado, por lo que se considera que no se ha producido detección.

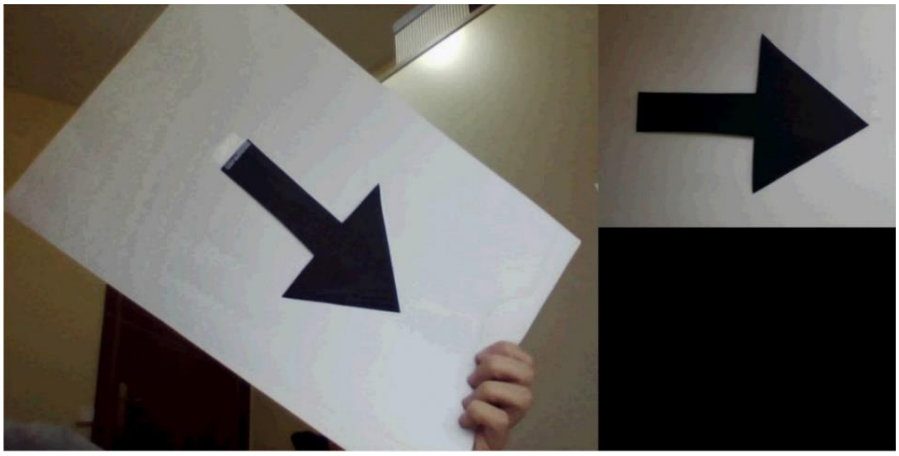


Figura 8 Match Template: Objeto rotado con respecto a la plantilla

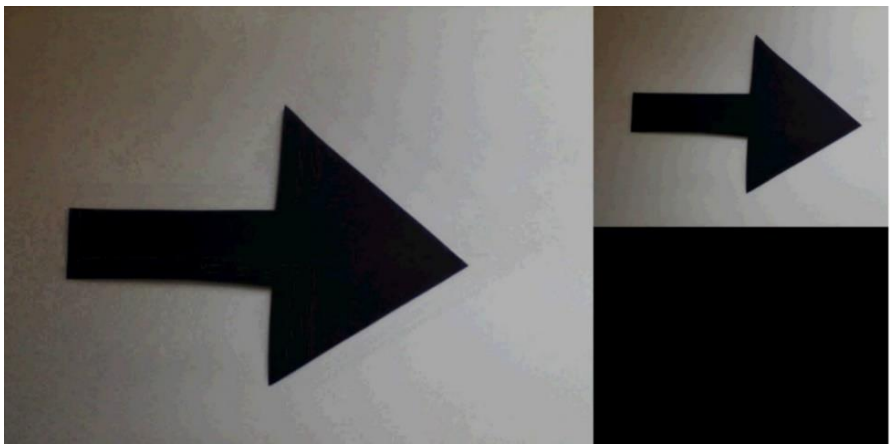


Figura 9. Match Template: Objeto a una escala mayor que la plantilla

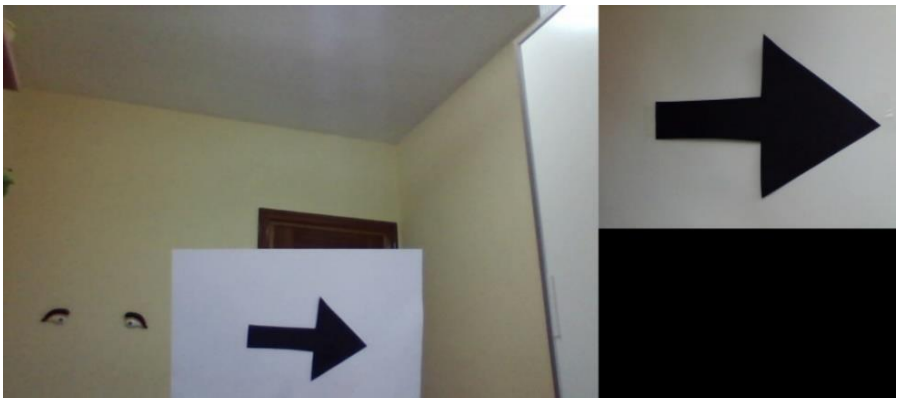


Figura 10. Match Template: Objeto a una escala menor que la plantilla

Además éste método tampoco es versátil en cuanto a la posibilidad de detectar un mismo objeto con una forma diferente *ver Figura 11,* es decir, si en la plantilla se tiene un objeto que es una flecha con una determinada forma y tamaño, y se desea detectar el mismo objeto (una flecha) pero con otra forma diferente tampoco habrá coincidencia entre la plantilla y el plano captado por la cámara, y no se producirá detección.

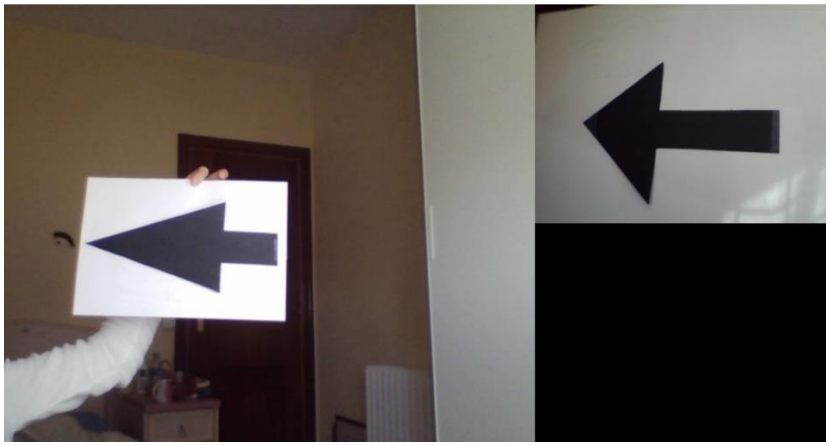


Figura 11. Match Template: Mismo objeto pero de forma diferente a la plantilla

En conclusión este método además de no ser invariante ni a la rotación ni a la escala sólo permite detectar un tipo de flecha con una determinada forma, la que esté en la plantilla.

2.1.2 Clasificadores

Para ser capaces de detectar objetos con un método que sea invariante a la rotación y escala, se suelen utilizar clasificadores. Existen numerosos tipos de clasificadores, pero en este apartado se va a explicar el basado en Haar para dar una idea de cómo funcionan. Se ha elegido éste en concreto porque viene incluido en la biblioteca de libre acceso de OpenCV, hermana de EmguCV que como se verá más adelante son las que se utilizan para implementar la solución final.

Antes de que un clasificador sea capaz de realizar la detección y reconocimiento de objetos es necesario entrenarlo, con el objetivo de “enseñarle” cuál es el objeto que se quiere reconocer en la imagen captada por la cámara. Para entrenarlo se utiliza una base de datos con cientos de imágenes que contengan el objeto que se quiere detectar desde diferentes ángulos y con diferentes condiciones de luminosidad, lo que se conoce como muestras positivas, y también se usan cientos de imágenes que no contienen el objeto, denominadas muestras negativas. Es importante que todas las imágenes utilizadas en el entrenamiento tengan el mismo tamaño.

Una vez que se ha efectuado el entrenamiento, se puede aplicar el clasificador a una región de interés del plano de la imagen captada por la cámara (donde se quiere buscar el objeto), es importante que esta región de interés sea del mismo tamaño que el de las muestras que se han utilizado en el entrenamiento. El clasificador devolverá un “1” si esa región tiene una alta probabilidad de contener el objeto buscado, y un “0” en caso contrario. Si no se tiene una región de interés y se quiere buscar el objeto en todo el plano captado por la cámara, se puede ir recorriéndolo e ir comprobando si la zona que se está recorriendo tiene probabilidad alta de contener el objeto. Por otro lado, si se quieren detectar objetos que tengan un tamaño diferente al tamaño de las muestras que se utilizaron en el entrenamiento (invariabilidad de escala), el clasificador está diseñado para ser redimensionado fácilmente, por lo que para encontrar el objeto de tamaño desconocido bastaría con recorrer el plano de la imagen captado por la cámara varias veces con diferentes escalas. De la misma manera, si se desea detectar objetos con una orientación diferente al de las muestras, también habrá de recorrerse varias veces el plano con diferentes orientaciones.

En general todos los clasificadores suelen ser del tipo “cascada”, lo que significa que están compuestos por clasificadores más básicos que se van aplicando por etapas. Para dar una detección por buena se han aplicado varias etapas sencillas previamente. Si la zona que se ha analizado ha pasado todas las etapas correctamente tendrá una alta probabilidad de contener el objeto buscado, mientras que si no superó alguna de ellas su probabilidad será muy baja. La utilidad que tiene el método de cascada es ahorrar el esfuerzo de procesamiento que supondría aplicar todas las etapas a una zona que se ha detectado desde el principio que tiene probabilidad muy baja de contener el objeto.

Las características tipo Haar son la entrada a los clasificadores más básicos. Es decir, la respuesta a cada etapa será calculada en función de las características de Haar que se muestran en la *Figura 12*:

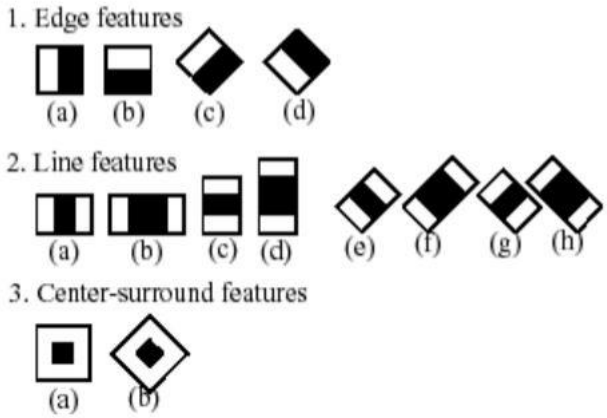


Figura 12. Características tipo Haar [15]

Por ejemplo en el caso de la característica 2d en la *Figura 12*, se calcula la diferencia entre la suma de todos los píxeles contenidos dentro del rectángulo mayor (el que contiene los dos rectángulos blancos y el negro) con respecto a la suma de los píxeles bajo el rectángulo negro multiplicado por tres para compensar las diferencias de tamaño entre las áreas. En función del valor resultante de efectuar esta operación se decidirá si la zona analizada cumple los requisitos para superar la etapa o no [23].

Inconvenientes:

Este clasificador no es capaz de detectar objetos con un amplio rango de orientaciones y escalas. Esto es debido al propio método que se utiliza para detectar objetos rotados o de diferentes tamaños. El método más sencillo es el que se ha explicado en este apartado:

1. Se entrena una única cascada con un determinado tamaño y orientación de muestras para después recorrer el plano varias veces modificando los tamaños y orientaciones de esas muestras.

Aunque hay otra posibilidad para abordar este aspecto que es:

2. Entrenar varias cascadas, cada una para detectar objetos en diferentes orientaciones y tamaños. Para después aplicar todas las cascadas a la zona de estudio donde se pretende encontrar el objeto.

Cualquiera de estos dos métodos finalmente tiene que ser unido en un único conjunto de detección, ya que las diferentes orientaciones y tamaños corresponden a un único objeto.

El principal problema es que el tiempo de clasificación empleado al usar cualquiera de estos métodos se multiplica cuantas más orientaciones y tamaños quieran contemplarse. En el caso de usar el segundo método el propio clasificador tiene la capacidad de decidir cuál de las cascadas es mejor para cada zona estudiada, consiguiendo reducir el tiempo de cómputo, pero perdiendo precisión al reducir este tiempo. Cabe destacar también que cuanto más se modifique la orientación y tamaño de las muestras de entrenamiento más se compromete el buen funcionamiento del clasificador.

Otro de los problemas que se presentan es que las orientaciones del objeto que el clasificador es capaz de detectar son limitadas, ya que el rango de diferencia de orientación entre las muestras no puede ser menor a 15º entre unas y otras para no comprometer el buen funcionamiento del clasificador.

Por último, por cada objeto diferente que se quiera detectar, se deberá entrenar un clasificador distinto. Por lo que el número de tipos de flecha a detectar también quedarían limitados empleando este método [11].

2.1.3 SURF y SIFT

SURF y SIFT también son algoritmos incluidos en las bibliotecas de libre acceso OpenCV y EmguCV. SURF es un método de detección de objetos invariante a la rotación y escala. Con una sola muestra del objeto que se quiere detectar es suficiente para poder encontrarlo en el plano que capta la imagen, indiferentemente de la orientación y escala con la que se presente.

SURF funciona de la siguiente manera:

Se tiene una imagen a la que se denominará “plantilla” que contiene únicamente el objeto que se va a buscar. El primer paso consiste en extraer los puntos de interés tanto de la plantilla como de cada frame de la imagen captada por la cámara. Los puntos de interés son aquellos puntos de la imagen que definen lo que es interesante o lo que destaca de la misma, y se calculan mediante métodos matemáticos que utilizan matrices hessianas. Al extraer los puntos de interés se obtiene información sobre su posición y el área que cubren en la imagen, pero sólo con esta información no se podrá saber si el objeto de la plantilla está trasladado, escalado o rotado en el plano que capta la cámara. Para obtener esta información es necesario calcular los descriptores que tienen asociados. Cada punto de interés que se ha encontrado en la imagen tiene un descriptor que lo acompaña. Los descriptores son una manera de comparar los puntos de interés, resumen en forma de vector alguna de las características de los mismos. Por ejemplo alguna de las características almacenadas podrían ser la intensidad en la dirección o la orientación más pronunciada.

A continuación se hace un “matching” para encontrar las coincidencias entre los puntos de interés de la plantilla con los de la imagen que capta la cámara. Para poder encontrarlas se estudian las características de los píxeles vecinos de cada punto de interés. Tras un análisis de las mismas se irán comparando y encontrando las coincidencias entre ambas imágenes.

SIFT funciona de forma muy parecida a SURF, de hecho este último surgió como una mejora de SIFT que es capaz de funcionar a mayor velocidad [24].

En la *Figura 13* se muestra un ejemplo de la detección con SURF. La imagen de la derecha es la plantilla que contiene el objeto que se quiere buscar, y la imagen de la izquierda es el plano de la imagen donde se quiere encontrar ese objeto. Los puntos azules representan los puntos de interés, y las líneas verdes las coincidencias que se han encontrado entre los puntos de la plantilla y la imagen que capta la cámara. El recuadro rojo que rodea el objeto en la imagen que capta la cámara indica que se ha efectuado la detección y el reconocimiento correctamente.

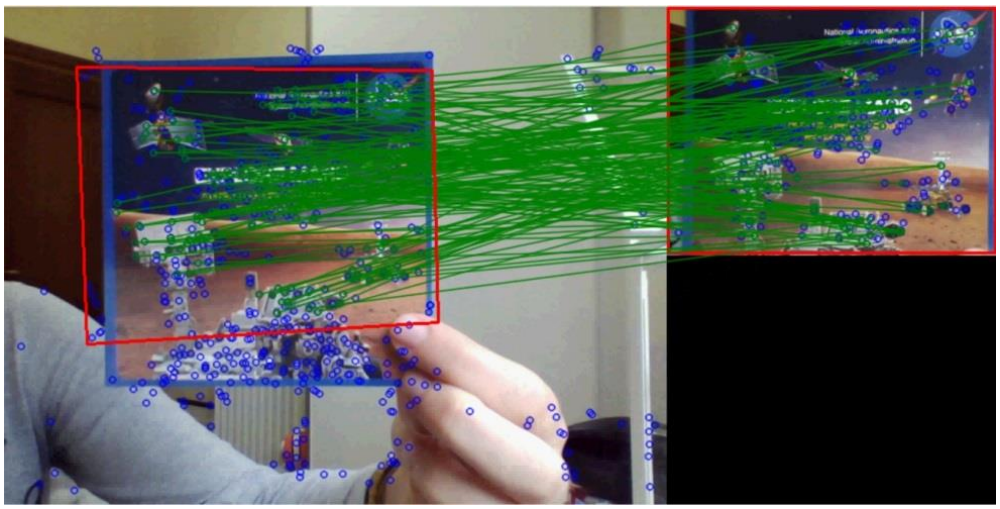


Figura 13. SURF: Detección de objeto

En las *Figuras 14 y 15* se muestran ejemplos del objeto a una escala mayor y menor que la de la plantilla. Aunque en el caso del objeto a escala menor *Figura 15*, las coincidencias de los puntos de interés de la plantilla y la imagen que capta la cámara que se han encontrado han sido menores, son suficientes para que la detección se produzca correctamente.

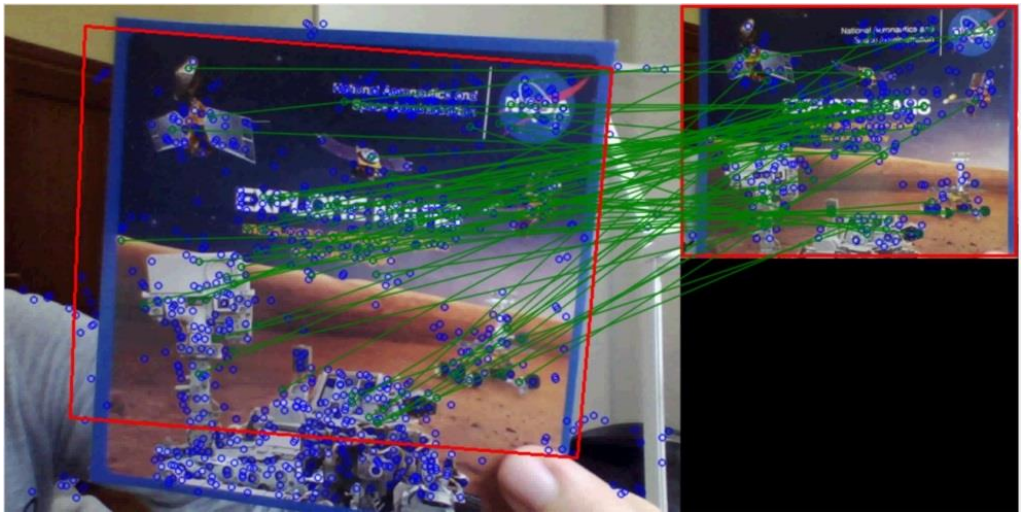


Figura 14. SURF: Objeto a mayor escala que la plantilla

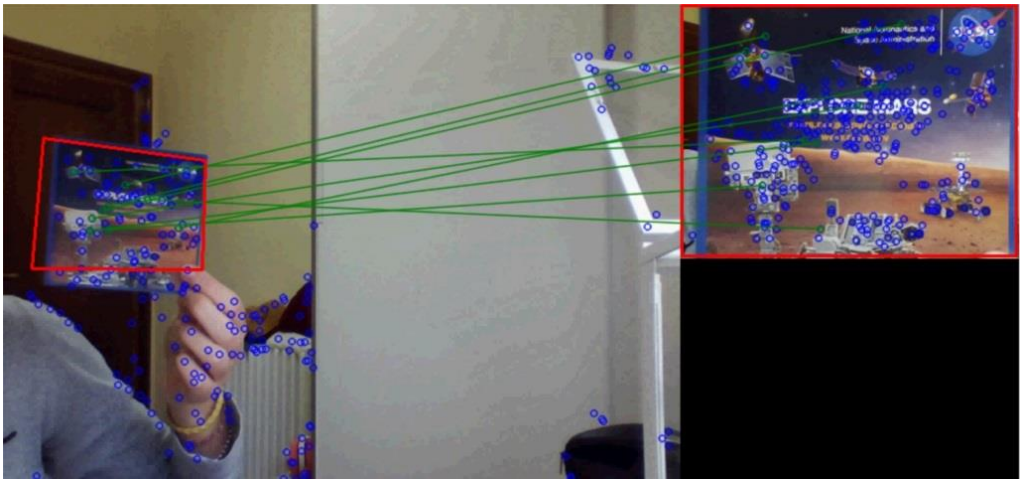


Figura 15. SURF: Objeto a menor escala que la plantilla

En la *Figura 16* se muestra el ejemplo de que el objeto esté rotado con respecto a la plantilla, como se puede observar la detección se produce correctamente.

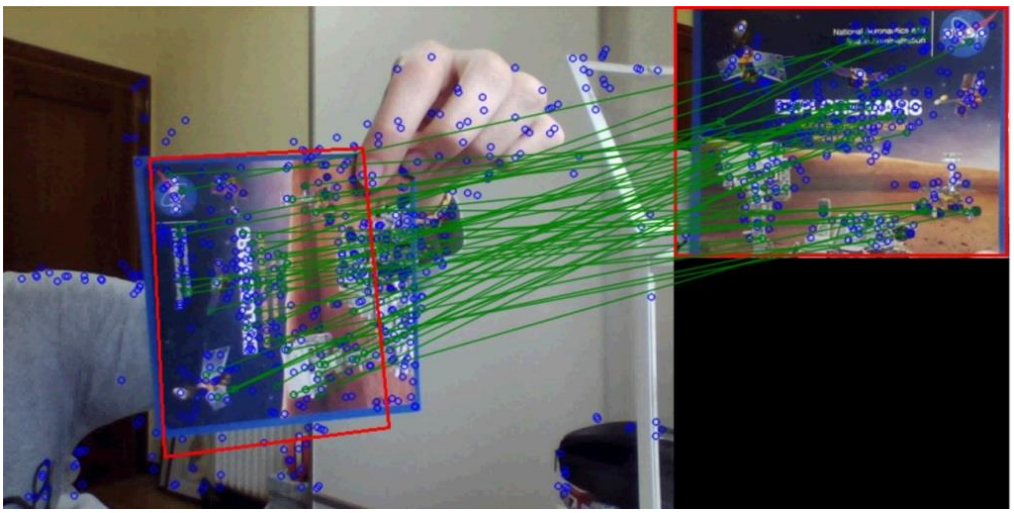


Figura 16. SURF: Objeto rotado con respecto a la plantilla

Inconvenientes:

SURF funciona bien con objetos a detectar que sean complejos, como en el caso de los ejemplos de las *Figuras 13, 14, 15* y *16,* ya que de ellos puede extraer una gran cantidad de puntos de interés, realizar un buen "matching" y tener la suficiente información para dar por válida la detección. Sin embargo en el caso de los objetos que se quieren detectar en este trabajo, flechas planas bidimensionales y de color uniforme que no presentan textura como en el caso de la *Figura 17,* se encuentra poca cantidad de puntos de interés y el "matching" es pobre. A veces esto es suficiente para dar la detección por buena, pero otras como en la *Figura 18*, sobre todo si el objeto se encuentra rotado o a una orientación diferente con respecto a la plantilla, no es suficiente y no se produce detección.

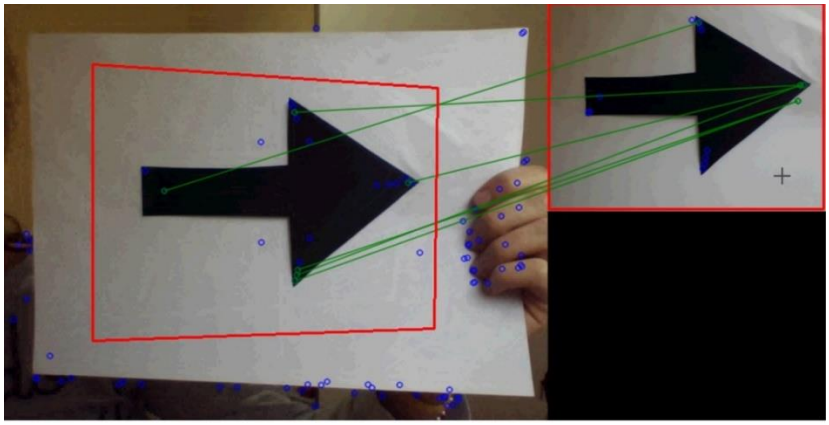


Figura 17. SURF: Detección de flecha

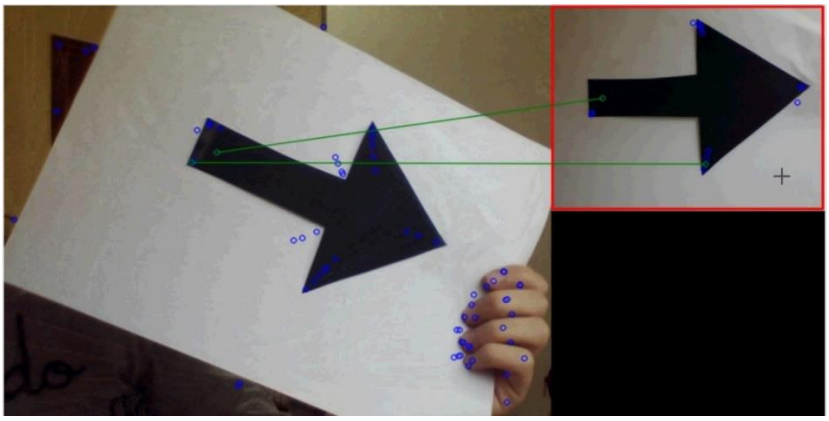


Figura 18. SURF: Flecha girada con respecto a la plantilla

Otro de los inconvenientes de SURF es que al utilizar únicamente una plantilla sólo es capaz de detectar un tipo de flecha, el que venga reflejado en la plantilla *ver Figura 19*. Si se quisieran detectar más tipos de flecha habría que crear una base de datos, lo que implica mayor esfuerzo y tiempo de procesamiento. Esto sería nefasto para la aplicación de este trabajo, en el que realizar estas operaciones en el menor tiempo posible es crucial debido a que el algoritmo se va a diseñar para emplearlo en competiciones (carreras de drones).

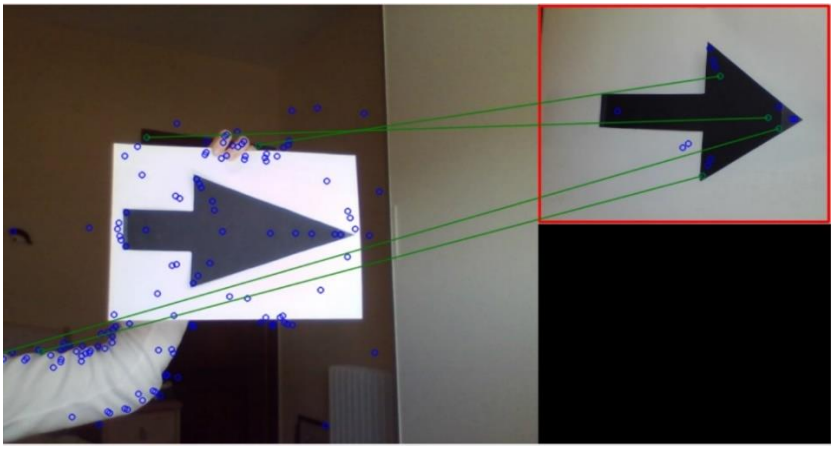


Figura 19. SURF: Mismo objeto pero de forma diferente al de la plantilla

SIFT es en algunos casos capaz de encontrar mayores puntos de interés y coincidencias en las imágenes que SURF, pero en este caso no supone una diferencia notable en la mejora de la detección.

2.1.4 Otros métodos

Existen otros métodos no incluidos en las librerías de OpenCV, sino que han sido desarrollados por investigadores que también podrían servir para el objetivo de este trabajo. Son los basados en la detección de forma, en este apartado se va a explicar uno en concreto [17].

Éste método se basa en la aproximación poligonal de la forma del objeto. Es invariante a la escala, la traslación y a algunos ocultamientos del contorno extraído. La idea principal en la que se basa es la de establecer que dos formas serán similares si empezando por un segmento de referencia y cubriendo una longitud determinada, se ha recorrido el mismo ángulo en ambas formas.

Es un algoritmo interesante porque toma un objeto como plantilla y puede encontrar aparte del objeto de la plantilla, el mismo objeto con variaciones en la forma. Por lo que en la aplicación de este trabajo podría encontrar varios tipos de flecha con una única plantilla, y por tanto sin demasiado esfuerzo de procesamiento. Sería un algoritmo rápido lo cuál para la aplicación de este trabajo, competir en carreras de drones autónomos es un aspecto fundamental.

Inconvenientes:

Uno de los inconvenientes es que la invariancia ante la rotación no está garantizada ya que es un método muy dependiente de la elección del primer segmento a partir del cual realizará el cálculo explicado. Además no está diseñado para trabajar a tiempo real.

2.2 Comparación

En este apartado se va a resumir cómo abordan los métodos expuestos en este capítulo las características básicas que se necesitan para resolver la problemática planteada *apartado 1.2* y conseguir el objetivo de este trabajo *apartado 1.3*: que un dron sea capaz de seguir un camino formado por flechas de diferentes tamaños y formas de manera autónoma en el menor tiempo posible y a tiempo real.

Las características básicas son las siguientes:

1. Detección de objetos de diferentes tamaños (invariabilidad a la escala)
2. Detección de objetos en diferentes orientaciones (invariabilidad a la rotación)
3. Detección de un mismo objeto con variación en su forma (detección de varios tipos de flecha)
4. Bajo tiempo de procesamiento
5. Trabajar a tiempo real

En cuanto a la primera característica, tiene doble finalidad, la primera es cumplir una de las bases que se establecen en la prueba *ver apartado 1.2.* Y la segunda es permitir que el dron pueda volar a diferentes alturas sin que se pierda la detección de los objetos.

Tabla 2. Comparación de los diferentes métodos

Métodos	Invariabilidad a la escala	Invariabilidad a la rotación	Detección de un mismo objeto con variación en su forma	Velocidad de procesamiento	Trabaja a tiempo real
Template Matching	No	No	No con una única plantilla, sí con base de datos	Rápida	Sí
Clasificadores	Sí (con esfuerzo de procesamiento)	Sí (con esfuerzo de procesamiento)	Sí (entrenando múltiples clasificadores)	Lenta	Sí
SURF/SIFT	Sí (muy pobre con objetos simples)	Sí (muy pobre con objetos simples)	No con una única plantilla, sí con base de datos	Rápida con una única plantilla, lenta con base de datos	Sí
Otros métodos	Sí	Sí (no garantizada)	Sí	Rápida	No

En la *Tabla 2* se puede observar que en general ninguno de los métodos expuestos cumple las cinco características básicas de una manera lo suficientemente óptima como para que se pueda resolver la problemática y cumplir el objetivo de este trabajo. En el caso de los tres primeros puede ser debido a que están pensados para que puedan detectar y reconocer objetos bastante complejos. Algunas de las aplicaciones más comunes en las que se usan son en la detección y reconocimiento de caras humanas y señales de tráfico.

Los clasificadores y SURF podrían llegar a detectar diferentes tipos de flecha en distintas orientaciones y escalas pero de forma precaria y lenta ya que no funcionan bien con objetos simples y sin textura como las flechas. Además necesitarían emplear mucho esfuerzo en operar con las bases de datos. Emplear bases de datos permite captar diferentes tipos de flechas, pero ese número siempre estará limitado por el número de plantillas introducido.

En este trabajo únicamente se quieren detectar flechas caracterizadas por ser objetos bidimensionales de color sólido que quedan totalmente definidas por su forma. Por esta razón es más lógico descartar los métodos anteriores y emplear en su lugar métodos que se centren en analizar la forma de los objetos detectados, determinando si cumplen las características que definen el objeto buscado. Empleando estos métodos no se tendrán limitaciones en cuanto a los tipos de flecha que se pueden detectar, ya que al no usar bases de datos se eliminan las limitaciones y además se consigue una velocidad de procesamiento mucho mayor.

El último método explicado en este capítulo tiene una filosofía que se adecúa mucho más al objetivo de este trabajo, ya que para analizar el objeto se centra en su forma, aunque no es capaz de funcionar a tiempo real y la invariabilidad ante la rotación es pobre.

Por todo ello en este trabajo se propone un algoritmo basado en el análisis de la forma de la flecha, que no va a emplear bases de datos. Además como se verá en el *Capítulo 5,* el algoritmo propuesto cumple las cinco características básicas necesarias para la consecución del objetivo de este trabajo de forma mucho más óptima.

Capítulo 3: Descripción general del sistema

3.1 Introducción a la aplicación

Este proyecto está formado por elementos de hardware y software. El hardware se compone del propio dron equipado con cámara desde la cual se captarán las flechas del recorrido, y el ordenador donde se va a ejecutar el programa. Dentro del software se contempla el lenguaje de programación (C#), las librerías de visión por computador (EmguCV), el entorno de trabajo (Microsoft Visual Studio), y la interfaz del programa.

Como se explicó en el *apartado 1.2* este trabajo está orientado a superar una prueba de una competición de drones autónomos. Dependiendo de la competición hay dos posibilidades, la primera de ellas es que el dron lo aporte la propia organización con la finalidad de que todos los participantes compitan en igualdad de condiciones, todos son drones iguales con las mismas características, de esta manera se evalúa únicamente el mejor algoritmo desarrollado. La segunda opción es que cada participante construya el suyo propio con la finalidad de que la prueba la gane no sólo aquel que haya desarrollado el mejor algoritmo, sino también el que haya sido capaz de construir el dron más ligero y veloz.

Figura 20. Drones de competición preparados para despegar [14]

3.2 Hardware

Este trabajo se centra únicamente en el desarrollo del algoritmo, que se puede implementar en cualquier dron. Es por ello que para la realización de las pruebas la elección del dron es indiferente, solamente es necesario que cumpla los siguientes requisitos básicos:

- Debe estar equipado con cámara orientada u orientable hacia el suelo para poder percibir las flechas.
- Debe llevar incorporado un magnetómetro, popularmente conocido como brújula (la justificación de esta necesidad se verá en el *Capítulo 4).*

- Debido a que la competición se puede desarrollar tanto en exterior como en interior, el dron elegido deberá tener un tamaño y rango de velocidades adecuado para que se pueda volar en interiores.

No se necesita por tanto un dron con características especiales, ya que la solución que se aporta en este trabajo podría implementarse en cualquiera. Para la realización de las pruebas del algoritmo se ha elegido el AR.Drone 2.0. de la marca Parrot, porque además de cumplir los requisitos expuestos, tiene una buena relación calidad – precio *ver Figura 21*.

Figura 21. AR.Drone 2.0 de la marca Parrot [22]

A continuación se detallan sus características técnicas. Aquellas que demuestran que se cumplen los requisitos especificados aparecen subrayadas:

Especificaciones técnicas:

- Máxima velocidad: 11,1 m/s
- Dimensiones: 77,7 x 38,3 x 12,5 mm
- Peso: 31g
- Precisión: +/- 2 metros
- Frecuencia: 5 Hz
- Voltaje: 5V
- Memoria flash: 4GB
- Batería recargable de Li-Po de tres elementos 1000 mAh.

Electrónica

- Procesador de 1 GHz y 32 bit ARM Cortex A8 con vídeo DSP TMS320DMC64x de 800 MHz
- Linux 2.6.32
- RAM DDR2 de 1GB a 200MHz
- USB 2.0 de alta velocidad para extensiones
- Wi-Fi b g n
- Giroscopio de 3 ejes con una precisión de 2000º/seg
- Acelerómetro de 3 ejes con una precisión de +/- 50mg
- Magnetómetro de 3 ejes con una precisión de 6º
- Sensor de presión con una precisión de +/- 10 Pa
- Sensores de ultrasonido para medir la altitud de avance

- Cámara vertical QVGA de 60 FPS para medir la velocidad de avance

Motores

- 4 motores de rotor interno “inrunner” sin escobillas. 14,5W 28500 RPM cuando queda suspendido en el aire.
- 1 cojinete de microbola (rodamiento de bolas en miniatura)
- 1 engranaje de Nylatron de bajo ruido para reductor de hélice de 1/8,75
- 1 eje de transmisión de acero templado
- 1 cojinete de bronce autolubricante
- 1 resistencia aerodinámica específica de alta propulsión.
- 1 CPU AVR de 8 MIPS por controlador de motor
- Parada de emergencia controlada por software
- Controlador de motor totalmente reprogramable
- Controlador electrónico del motor resistente al agua

Vídeo

- Cámara HD 720p 1280x720 píxeles 30 FPS
- Lente gran angular: Diagonal 92º
- Perfil base de codificación H264
- Transmisión de baja latencia
- Almacenamiento de vídeos durante el vuelo con el dispositivo remoto
- Foto JPEG
- Almacenamiento instantáneo de vídeos con Wi-Fi, directamente en el dispositivo remoto o en una memoria USB

3.3 Software

3.3.1 Lenguaje de programación

El lenguaje de programación elegido para el desarrollo del algoritmo que se propone en este trabajo ha sido C#.

C# es un lenguaje dirigido principalmente a los desarrolladores que crean aplicaciones utilizando el framework .NET.

La sintaxis que utiliza C# es muy expresiva, y puede ser reconocida por cualquier persona familiarizada con C, C++ o java. La ventaja que tiene C# es que simplifica varias de las complejidades del lenguaje C++ y además proporciona características tales como el acceso directo a memoria, tipos de valor que admiten NULL, enumeraciones etc. que no son posibles de encontrar en Java [18].

3.3.2 Librerías de visión por computador

OpenCV es una biblioteca libre de procesamiento de imagen. Estas bibliotecas surgen con la idea de facilitar el trabajo a los desarrolladores, y ahorrarles la tarea de definir desde cero cada operación que se quiera aplicar a las imágenes.

OpenCV contiene más de quinientas funciones programadas en lenguaje C y C++ de diferentes ramas dentro del ámbito de la visión por computador. Sin embargo no es tan amigable con las aplicaciones que se desarrollan utilizando el framework .NET, ya que

desde el sistema .NET no es posible llamar a funciones y métodos programados en C y C++. Para resolver este inconveniente surgió EmguCV.

Se podría decir que EmguCV es lo mismo que OpenCV pero en .NET. Además de abarcar diferentes áreas en visión como calibración de la cámara, visión estéreo o robótica también es capaz de invocar funciones y métodos de la librería OpenCV.

Por estas razones, y como esta aplicación se va a desarrollar en lenguaje C# *ver apartado 3.3.1* la librería de visión por computador que se va a utilizar en el desarrollo del algoritmo propuesto en este trabajo es EmguCV [19].

3.3.3 Entorno de programación

El algoritmo se va a desarrollar en forma de aplicación con una interfaz de usuario que podrá ejecutarse localmente en los equipos de los usuarios. Para ello se va a elegir el entorno de programación Microsoft Visual Studio.

Visual Studio es un entorno de desarrollo integrado IDE para sistemas operativos Windows. Soporta gran variedad de lenguajes de programación (C, C++, Visual Basic .NET, Java, Python etc.) y también entornos de desarrollo web.

Visual Studio permite desarrollar el algoritmo en forma de aplicación, configurando el proyecto en Visual C# y escogiendo la opción de Windows Forms Applications. Gracias a ello aparte de crear la aplicación, también se va a poder diseñar la interfaz de usuario. La interfaz del algoritmo que se propone en este trabajo se muestra en la *Figura 22.*

Figura 22. Interfaz de la aplicación

Capítulo 4:

Detección y reconocimiento de marcadores para el guiado de un UAV

4.1 Introducción

En este apartado se va a explicar de forma detallada el algoritmo propuesto como solución a la problemática expuesta en el *apartado 1.2,* y que cubre los objetivos marcados en el *apartado 1.3.*

Primero se incluye el diagrama de estados y un algoritmo en pseudo-código para facilitar al lector la comprensión de las explicaciones aportadas en este capítulo. A continuación le sigue la explicación del algoritmo solución dividida en cuatro grandes bloques: segmentación, etapa previa con la finalidad de empezar a separar el objeto a detectar del resto de elementos que contiene la imagen, detección, etapa en la que se hace al algoritmo consciente de los objetos que contiene, reconocimiento gracias al cual el algoritmo es capaz de distinguir qué objetos son flechas y cuáles no lo son, y guiado en el que se va a transmitir toda la información obtenida en las diferentes etapas del algoritmo al controlador del dron para que efectúe los movimientos correspondientes.

Durante la explicación de las diferentes etapas del algoritmo se nombran entre comillas y negrita métodos que se han utilizado pertenecientes a la librería de EmguCV. Además se trabaja con gran cantidad de abreviaturas cuyo significado se encuentra en el apartado “Abreviaturas” al comienzo de este trabajo.

Este capítulo finaliza con la exposición de las alternativas de diseño que surgieron a la hora de desarrollar la solución propuesta.

4.2 Diagrama de estados

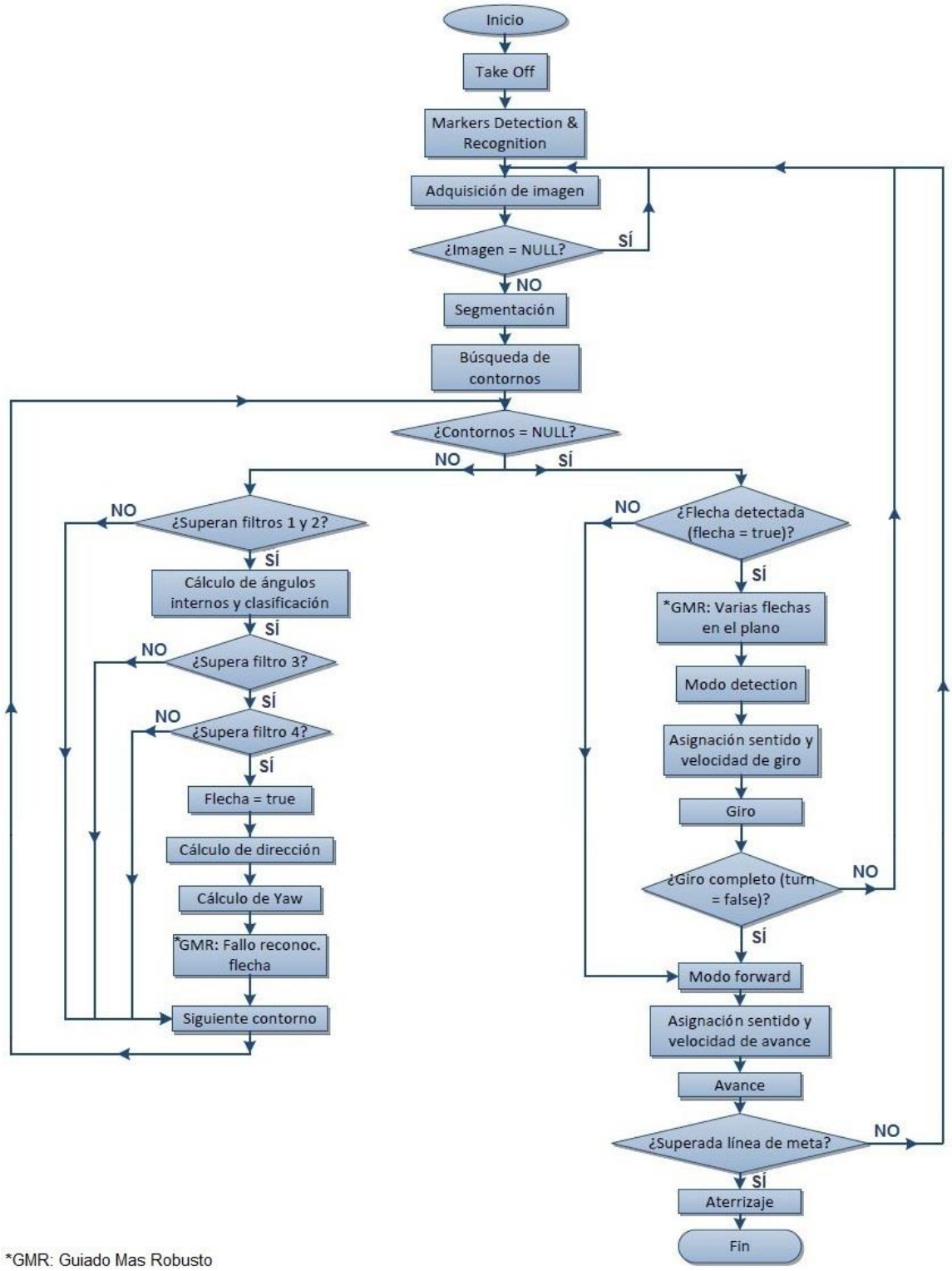


Figura 23. Diagrama de estados

4.3 Algoritmo

Algorithm: Markers Detection and Recognition For UAVs Visual Navigation Systems

Input: Image frame F_t
Output: Pitch Angle $Pitch$, Yaw Angle Yaw
Define: Arrow Detected $isArrow$, Yaw Angle Yaw, Direction Vertex Vector $VDvector$, Yaw Angle Vector $YawVector$, Compass Value $PsiAddition$, Turning $turn$, Turn angle $reference$, Previous Compass Value $PreviousPsi$, Addition $Addition$, Same Sign $YawNS$, Finish line $FLine$
while $isFlying()$ ***do***
    $F_t \leftarrow$ getNewFrame( )
    if $F_t \leftarrow isNotEmpty()$ **then**
        Define: Thresholded Image $Thresh$, Threshold T, Contours $contours$, Arrow Found $ArrowFound$, Frame Counter $temp$
        $Thresh \leftarrow$ ThresholdBinary(F_t, T)
        $contours \leftarrow$ FindContours(Thresh)
        while $contours \leftarrow isNotEmpty()$ **then**
            Define: Contours Approximation $contoursAp$, Minimum Area $minArea$, Vertex number $numVertex$, Arrow Back mid, Arrow Direction $direction$
            $contoursAp \leftarrow$ ApproxPoly(contours, 10.0)
            if $contoursAp \leftarrow Area() > minArea$ **then**
                if $contoursAp \leftarrow Total() = numVertex$ **then**
                    Define: Vertexes Array $points$, Direction Vertex $VD\ Arrow$, Sides $edges$, Acute Angles $acute$, Straight Angles $straight$, Acute Number $numAcute$, Straight Number $numStraight$, Previous Distance $PrevDistance$
                    $points \leftarrow$ ToArray(contoursAp)
                    $edges \leftarrow$ PolyLine(points, true)
                    $VD \leftarrow$ FindVD(points, PrevDistance)
                    $acute, straight \leftarrow$ CalculateAndClassify(VD, edges)
                    $numAcute \leftarrow$ CalculateNumberAcutes(acute)
                    $numStraight \leftarrow$ CalculateNumberStraight(straight)
                    if (1 < numAcute < 4) and (1 < numStraight < 5) **then**
                        $isArrow \leftarrow$ ParallelCondition(VD, points)
                        if $isArrow\ is\ true$ **then**
                            $ArrowFound \leftarrow$ true
                            $mid \leftarrow$ CalculateMidPoint(points, VD, edges)
                            $direction \leftarrow$ CalculateArrowDirection(VD, mid)
                            $VDvector \leftarrow$ Add(VD)
                            Define: Vertical begin $beginV$, Vertical end $endV$, Vertical Line $vertical$
                            $vertical \leftarrow$ CalculateVertical(beginV, endV)
                            $Yaw \leftarrow$ CalcuateYaw(direction, vertical)
                            $YawVector \leftarrow$ Add(Yaw)
                        end
                    end
                end
                $contours \leftarrow Next()$
            end
            Define: Vector Position $index$, Previous Upper Vertex $PrevUVD$, Previous

```
        Arrow Position PrevAP
        index ← CalculateUpperArrow(VDvector, PrevUVD)
        VDvector[index], YawVector[index] ← MoreRobustMethods(index, PrevAP)
        Define: Spin velocity and direction yawValue,  Forward velocity and
        direction pitchValue
      if ArrowFound is true then
          turn ← true
          PsiAddition ← Psi( )
          reference ← Yaw
      end
      else
          if turn is true then
              if (PsiAddition > 0  and Psi( ) > 0)or(PsiAddition < 0 and Psi( ) < 0) then
                  Addition ← PsiAddition + reference
                  PreviousPsi ← Psi()
                  YawNS ← Addition − Psi( )
              end
              else
                  Addition ← YawNS − PreviousPsi
              end
              Yaw ← Addition − Psi( )
          end
          else
              Yaw ← 0
              pitchValue ← 0.1
              temp + +
          end
      end
      if FLine is true then
          Landing( )
      end
      else
          if Yaw > 0 then
              yawValue ← 0.5
          end
          else
              if Yaw < 0 then
                  yawValue ← −0.5
              end
              else
                  yawValue ← 0
              end
          end
          Navigate( 0,  pitchValue,  yawValue, 0)
      end

  end
end
```

4.4 Solución final

4.4.1 Segmentación

El primer paso que se implementa en la solución es realizar una segmentación, es decir, separar el objeto con el que se quiere trabajar, del resto de elementos de la imagen que no se van a analizar. De esta manera en los siguientes pasos de la solución propuesta se elimina en la medida de lo posible el coste computacional que supone analizar la imagen completa, y se puede focalizar el análisis en el objeto de interés, que en el caso de este trabajo son las flechas.

Para conseguir efectuar esta separación se binariza la imagen captada, con el fin de obtener una imagen con únicamente dos colores negro y blanco. El primero de ellos se le asignará al objeto a estudiar, y el otro, el blanco, se corresponderá con el resto de la imagen.

4.4.1.1 Obtención del umbral

Para poder efectuar la binarización se necesita establecer un umbral, que indicará el rango de valores entre los que varían las intensidades de los píxeles que forman la flecha. Después se compararán todos los píxeles de la imagen bajo la siguiente condición: si el valor de la intensidad del píxel analizado se encuentra dentro del umbral establecido, se le asigna valor 0 que es el correspondiente al color negro. Si por el contrario ese valor está fuera del umbral se le asigna valor 255, correspondiente al color blanco.

El siguiente paso es saber qué valor de umbral debe escogerse. La meta perseguida es conseguir obtener una imagen final con la flecha totalmente definida en píxeles negros, y el fondo en píxeles blancos. Para ello se necesita ir cambiando los valores del umbral hasta dar con el que permita obtener la imagen descrita. Este proceso podría resultar tedioso si se prueban los valores aleatoriamente, por lo que en su lugar se utiliza una barra de valores, cuyo cursor se puede ir moviendo a tiempo real permitiendo observar más rápidamente los cambios que producen en la imagen los diferentes valores de umbral, que vienen dados por el valor que indica de la barra en cada momento.

El proceso detallado es el siguiente: Se parte de la imagen original *ver Figura 24,* a la que se le aplica la transformación a binario con la función **“ThresholdBinary()”.** Esta función necesita el input del valor del umbral pero como en este punto se está en proceso de averiguarlo, se le introduce como input el valor que vaya marcando la barra de valores de la que se ha hablado anteriormente.

Figura 24. Imagen original

Como se observa en la imagen superior izquierda de la *Figura 25,* se parte con un valor de umbral de 255, siendo la imagen totalmente negra, lo que indica que se está detectando absolutamente todo lo que está contenido en la imagen. La meta es detectar únicamente la flecha, por lo que es necesario reducir el umbral. En las siguientes imágenes de la *Figura 25* se observa cómo al reducirlo la flecha cada vez está más definida y el fondo cada vez va tornándose más blanco. Esto indica que la segmentación se está ejecutando correctamente, es decir, se está consiguiendo separar el objeto a analizar del resto de la imagen que no interesa.

El valor que debe seleccionarse finalmente, al que se le denominará umbral límite, es aquel que permita filtrar mejor el fondo, pero sin entrar en conflicto con la buena detección del objeto que se quiere segmentar. Esto se ilustra mejor en las dos últimas imágenes inferiores de la *Figura 25*: Con un valor de umbral igual a 77 se obtiene una segmentación casi perfecta (aún se observan algunos píxeles negros de fondo), pero con un umbral de 66 la flecha comienza a perder definición. Por tanto el umbral límite será un valor inferior a 77 para intentar eliminar los píxeles negros de fondo que aún se observan y mayor que 66 para no perder la buena definición de la flecha.

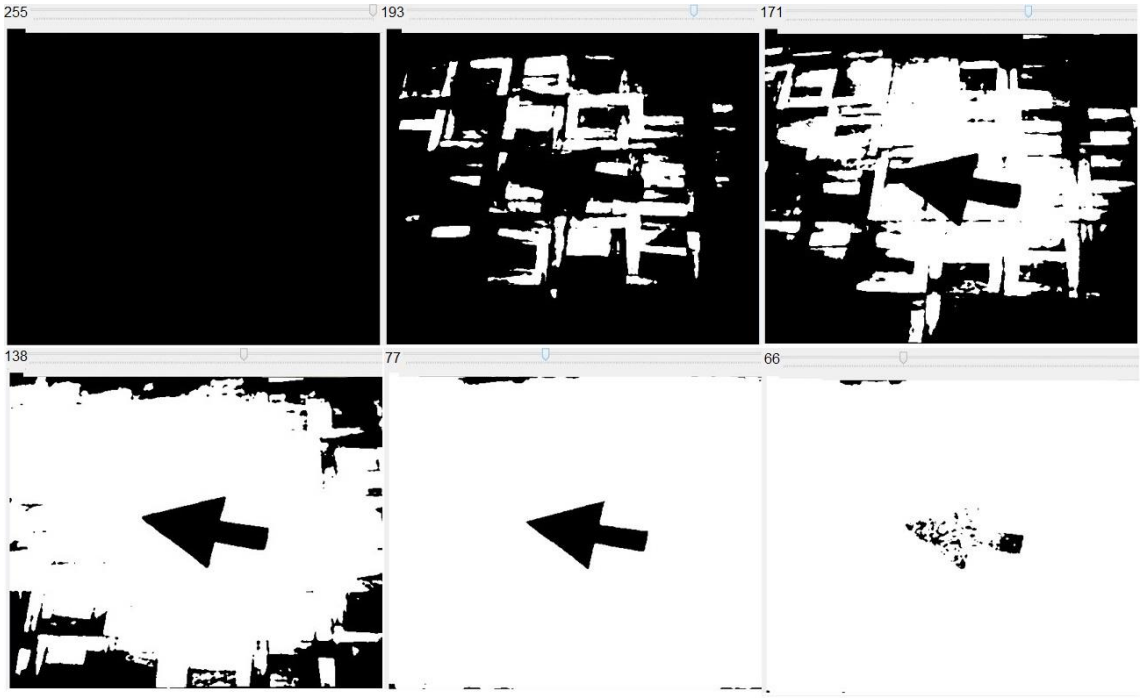


Figura 25. Proceso de selección de umbral

Hasta este punto se ha explicado el procedimiento seguido para la elección del umbral límite, pero con realizar estos pasos una única vez no es suficiente:

Las condiciones de luminosidad ambiental afectan en gran medida al valor del umbral límite. Si se desea que la solución final pueda implementarse contemplando el mayor rango de condiciones lumínicas posible, será necesario repetir varias veces el procedimiento citado anteriormente con diferentes niveles de luz. El valor del umbral final será cercano a un promedio de los valores umbral límite de todas las pruebas realizadas, escogiendo finalmente aquel que menos comprometa la correcta definición del objeto de interés en las diferentes condiciones de luminosidad ambiente.

Debido a que las condiciones de iluminación que pueden darse son infinitas, nunca se obtendrá un umbral ideal que permita hacer una segmentación perfecta. Contemplar la mayor cantidad de condiciones lumínicas posibles supone empobrecer la segmentación, ya que para no comprometer la buena definición de la flecha en todos los casos probados, se habrá sacrificado filtrar correctamente el fondo de la imagen. Al no filtrar adecuadamente el fondo se obtienen píxeles negros que no forman parte de las flechas, lo que puede acarrear problemas en siguientes fases del algoritmo tales como dificultar el reconocimiento de las flechas, o hacerlo susceptible a falsas detecciones. Habrá por tanto que emplear otros métodos, explicados en los siguientes apartados de esta memoria, para evitar en la medida de lo posible estos errores.

En conclusión realizar una buena segmentación es crucial para el correcto funcionamiento de la solución final. Para conseguirla es necesario elegir un valor de umbral adecuado, que permita detectar la flecha en diferentes condiciones lumínicas. Sin embargo cuanto mayor sea el rango de casos de luminosidad ambiente contemplados, peor será la segmentación realizada. Será necesario por tanto llegar a un punto de equilibrio que permita la detección de la flecha en un amplio rango de condiciones de luminosidad ambiente, subsanando los errores derivados de no tener una segmentación ideal con los siguientes pasos del algoritmo.

4.4.2 Detección

Una vez realizada la segmentación, se tiene separado el objeto del fondo, pero el algoritmo no es capaz de saber que en la imagen realmente hay un objeto puesto que no tiene ninguna información sobre lo que contiene. La detección consiste precisamente en esto, en hacer al algoritmo consciente de los objetos que contiene.

El siguiente paso es, por tanto, extraer características de la imagen para proveer de información al algoritmo sobre los objetos contenidos. Para ello lo que se va a hacer es buscar la forma que tiene cada uno de ellos, esto se consigue gracias a la extracción de los contornos. En este punto cabe destacar que la forma de un objeto viene definida por los contornos del mismo, por lo que hablar de forma es lo mismo que hablar de contornos.

4.4.2.1 Contornos

Los contornos son aquellas líneas que unen los píxeles de la frontera del objeto que tienen la misma intensidad. Son la forma del objeto, *ver Figura 26.*

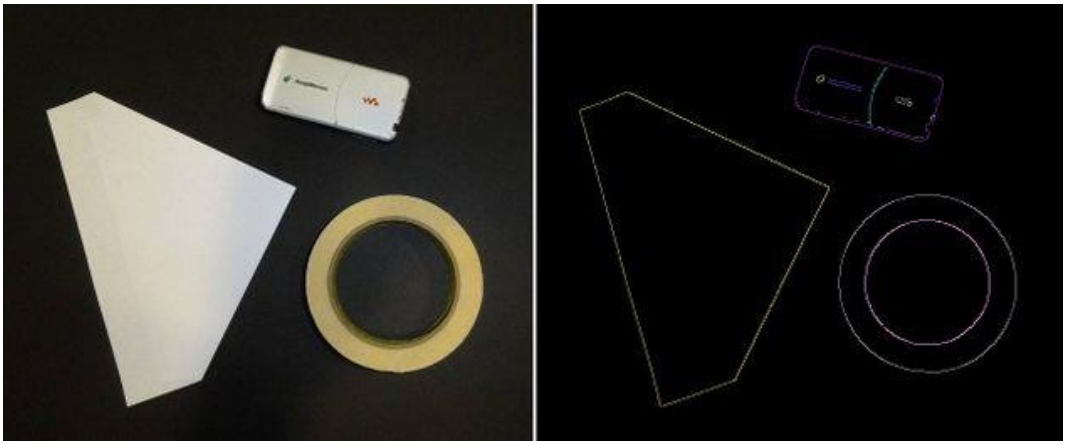


Figura 26. Contornos de varios objetos [5]

Para calcular el contorno de una línea recta no es necesario guardar todos los puntos que conforman esa línea, simplemente con almacenar los puntos de los extremos quedarán definidos. Esto puede entenderse mejor con el ejemplo de la *Figura 27*: en el triángulo de la izquierda se han guardado todos los puntos de la frontera que tienen la misma intensidad, mientras que en el triángulo de la derecha tan sólo se han guardado los extremos de cada una de las líneas rectas, que al ser el triángulo un polígono coincide con sus vértices. A la hora de dibujar los contornos, ambos triángulos tendrán la misma representación, con la diferencia de que para calcular y almacenar los contornos del triángulo de la izquierda se ocupa mucha más memoria que en el de la derecha.

Por otro lado, como el objeto de este trabajo es detectar y reconocer flechas, y éstas son polígonos cuyos lados son líneas rectas, se verá que en los siguientes apartados disponer de los vértices de la misma será de gran utilidad. Es por todas estas razones que se utiliza el método mostrado en el triángulo de la derecha de la *Figura 27* para calcular los contornos, al que se le denomina "método de aproximación simple".

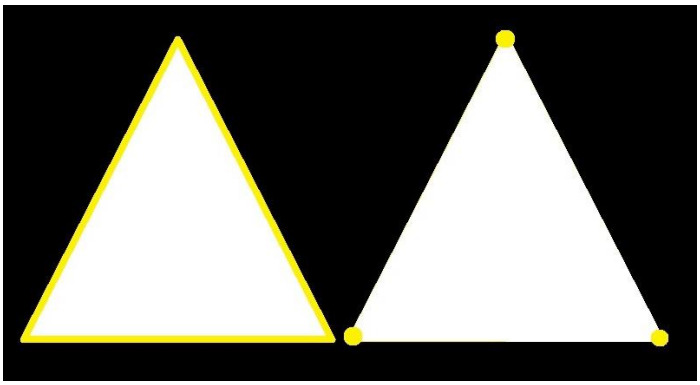


Figura 27. Formas de almacenar puntos de contornos

Una vez elegido el método para calcular los contornos se puede emplear la función **"`FindContours()`"** para buscarlos y extraerlos. Acto seguido, a los contornos encontrados se les aplica el método **"`ApproxPoly()`"** basado en el algoritmo de Ramer-Douglas-Peucker (RDP) [16] y [7]. Este algoritmo parte de un contorno original compuesto por los puntos correspondientes a los vértices de la forma, y busca encontrar un contorno similar pero compuesto de una menor cantidad puntos *ver Figura 28*. Esto se hace por varios motivos, el principal es ajustar la forma del objeto a la realidad, por ejemplo se trata de eliminar vértices detectados que no sean parte de la forma sino que se correspondan con ruido o hayan sido originados debido a una mala segmentación. Gracias a esta operación se consigue también simplificar el procesamiento.

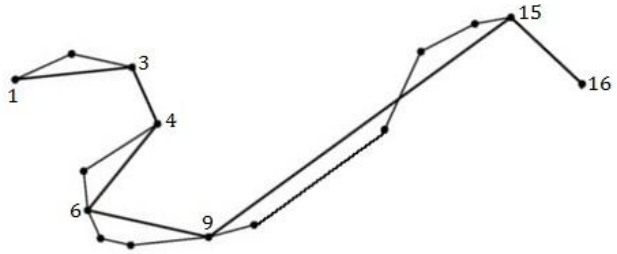


Figura 28. Ejemplo de funcionamiento del algoritmo Ramer-Douglas-Peucker (RDP)

Una vez realizadas las operaciones de extracción de contornos, el algoritmo ahora sí que tiene información sobre todos los objetos que contiene, es decir, es capaz de detectarlos gracias a que ha extraído su forma (contornos), por lo que se puede pasar a la fase de reconocimiento.

4.4.3 Reconocimiento

Cuando se tienen detectados todos los objetos que contiene la imagen, puede efectuarse el reconocimiento, es decir, discernir si los objetos captados por la cámara son en efecto una flecha.

Debido a que las flechas a detectar son objetos en dos dimensiones de un mismo color uniforme y sólido, la principal característica que poseen y que las distinguen de los demás objetos es su forma, detallada en el *apartado 4.4.3.1*. Por lo tanto para dar por buena que una detección es una flecha se establece que si ésta presenta todas las características que definen la forma de una flecha, el objeto será una flecha.

Debido a esto, la filosofía que se seguirá a la hora de efectuar el reconocimiento será utilizar los elementos característicos de la forma de la flecha para ir efectuando sucesivos filtrados. En cada uno de ellos se hará una comparativa en la que se considerará flecha a aquellos objetos que cumplan unas determinadas condiciones, y se descartarán los que no las cumplan, es decir los que no son flechas.

4.4.3.1 Forma

Una flecha es un polígono. Se define polígono como una figura geométrica plana limitada por tres o más rectas y que tiene tres o más ángulos.

Las flechas a detectar son objetos bidimensionales de color uniforme y sólido que quedan totalmente definidas por su forma. A continuación se van a indicar las características más importantes de la misma que las distinguen de otros objetos.

- **Numero de lados y vértices**:

  Todas las flechas están formadas por siete lados, como se observa en la *Figura 29.*

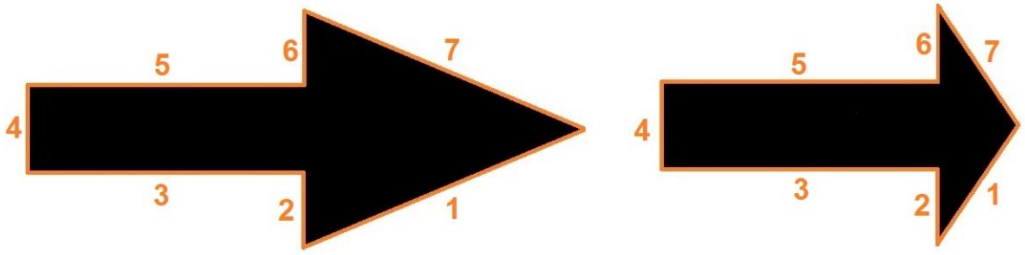


Figura 29. Número de lados de dos flechas

En los polígonos el número de vértices es coincidente con el número de lados del mismo, por lo que las flechas tendrán también siete vértices, *ver Figura 30.*

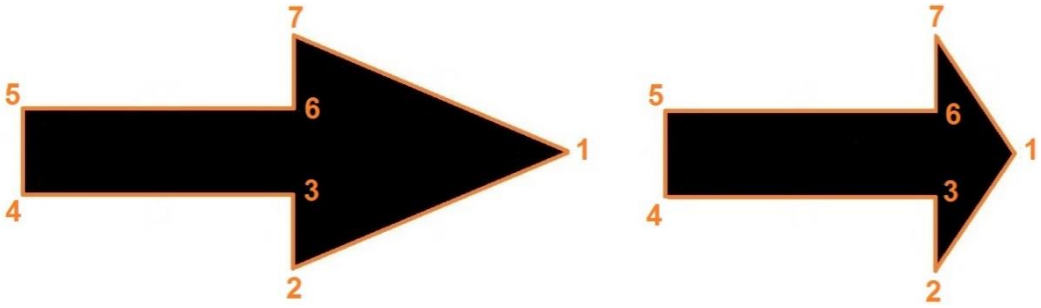


Figura 30. Número de vértices de dos flechas

- **Ángulos:**
  Para empezar a definir las características de los ángulos primero cabe observar que la forma de la flecha se podría dividir en dos subformas un rectángulo y un triángulo, *ver Figura 31.*

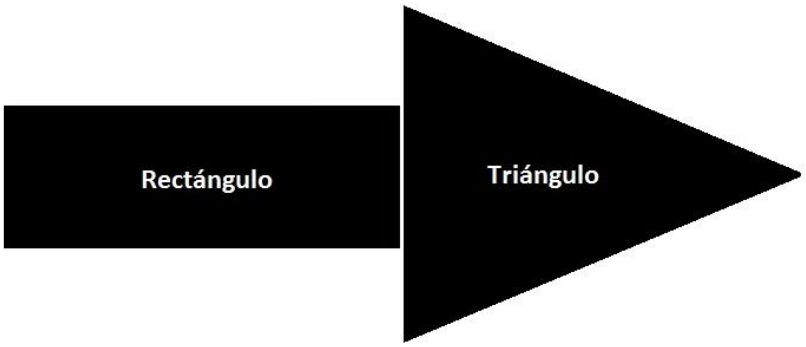


Figura 31. Flecha dividida en dos subformas (triángulo y rectángulo)

Como bien es sabido los rectángulos se caracterizan por estar formados por cuatro ángulos rectos de 90º. Por otro lado, los triángulos están formados por un total de tres ángulos, además según las reglas matemáticas se sabe que la suma de los tres ángulos de un triángulo tiene que ser igual a 180º.

Debido a esto, la parte final de la flecha cuya forma se asemeja a la de un rectángulo va a estar formada por cuatro ángulos rectos de 90º. A su vez, la punta de la flecha cuya forma se asemeja a la de un triángulo va a poder estar formada o bien por tres ángulos agudos, o por dos agudos y uno recto u obtuso v*er Figura 32.* En el caso del tipo de flechas mostradas en la *Figura 33,* la zona triángulo estará formada por tres ángulos agudos, y la zona rectángulo por dos rectos y dos agudos.

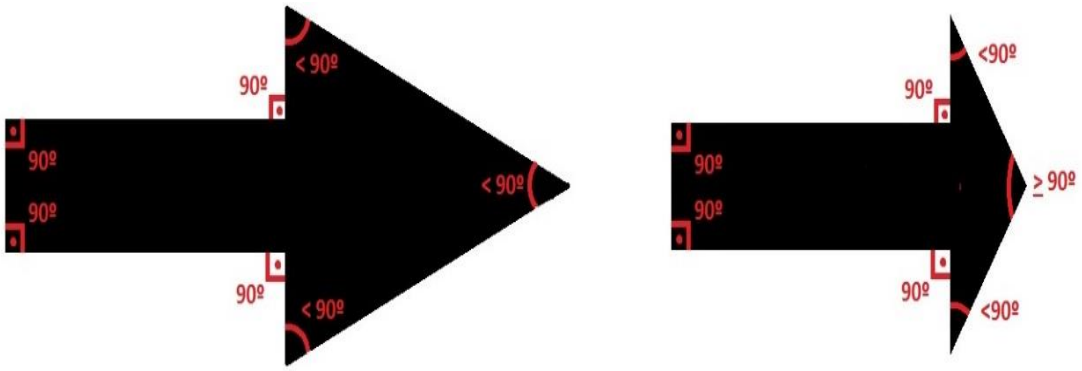


Figura 32. Ángulos en diferentes tipos de flechas (I)

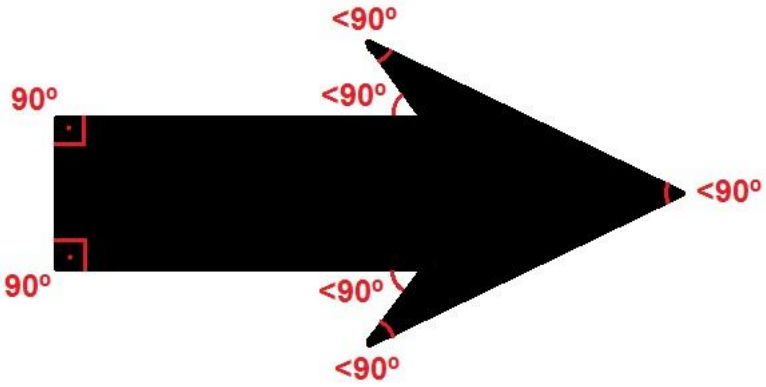


Figura 33. Ángulos en diferentes tipos de flecha (II)

- **Paralelismo:**
  Otra de las características que definen la flecha es el paralelismo de sus lados. En la *Figura 34* se puede observar que aquellos lados marcados del mismo color (en rojo o amarillo) son paralelos entre sí.

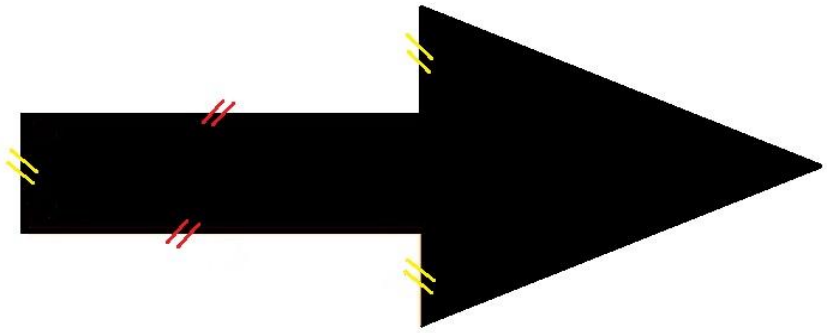


Figura 34. Lados paralelos de una flecha

4.4.3.2 Filtrados

Como se explicó en el *apartado 4.4.3*, la filosofía a seguir en esta fase del algoritmo va a ser utilizar las características de la forma de la flecha para efectuar sucesivos filtrados. En cada uno de ellos se van a descartar aquellas detecciones cuya forma no se corresponda con la de una flecha. En este trabajo se han implementado un total de cuatro filtrados: Filtro 1 Área mínima, Filtro 2 Número de vértices, Filtro 3 Número de ángulos de cada tipo y Filtro 4 Paralelismo.

Partiendo de la situación expuesta en el *apartado 4.4.2.1,* (se tiene una imagen a la que se le han buscado y extraído los contornos de cada uno de los objetos que contiene) se puede realizar el primer filtrado.

- **Filtro 1 Área mínima:**

La condición que se va a establecer para realizar la comparativa en el primer filtrado es la siguiente: Descartar aquellos objetos cuyo área sea demasiado pequeña como para ser una flecha. Además con esto se pretende descartar posibles detecciones que aparezcan debido a que la segmentación no sea demasiado buena (sombras, otros objetos etc).

- **Filtro 2 Número de vértices:**

El siguiente filtrado establece la condición de que entre todos los objetos presentes en la imagen, sólo podrán tener posibilidades de ser flecha aquellos compuestos por siete vértices *ver apartado 4.4.3.1*. Aquellos objetos formados por un número de vértices diferente no serán flecha, incumplirán la condición y se procederá a descartarlos.

Para traducir esto en el algoritmo, se utiliza el método **".Total"** que informa sobre el número de puntos que se han almacenado para calcular el contorno. Como se ha indicado en el *apartado 4.4.3.1* en el caso de los polígonos coincidirá con el número de vértices.

Establecer como condición de filtrado un número de puntos guardados para calcular los contornos igual a siete, implica que todas las detecciones que no se han descartado van a ser figuras formadas por líneas rectas. Esto es debido a que para representar contornos que no sean rectas es necesario guardar un mayor número de puntos característicos *ver Figura 35*, que en el caso de figuras formadas por líneas rectas en las que sólo se guardan los puntos extremos *ver Figura 36*.

Por todo lo expuesto anteriormente se puede afirmar que los contornos que hayan superado el segundo filtro serán figuras formadas por líneas rectas, de las que puede haber dos tipos, figuras cerradas llamadas polígonos como en el ejemplo de la parte izquierda de la *Figura 36,* o figuras abiertas como en el ejemplo de la parte derecha de la *Figura 36.* Para ambos casos se tendrán almacenados sus vértices.

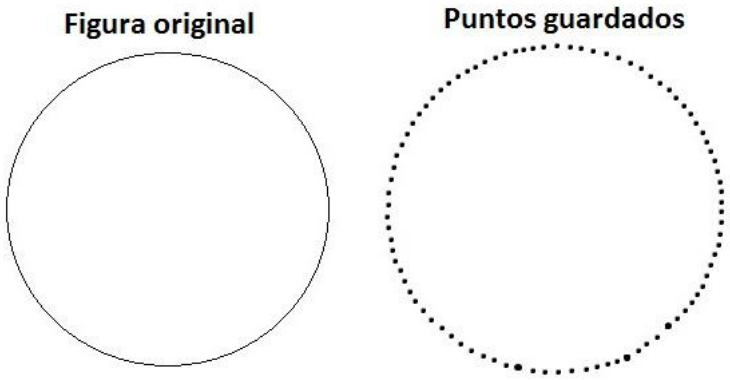


Figura 35. Original y puntos guardados para el cálculo del contorno de una figura circular

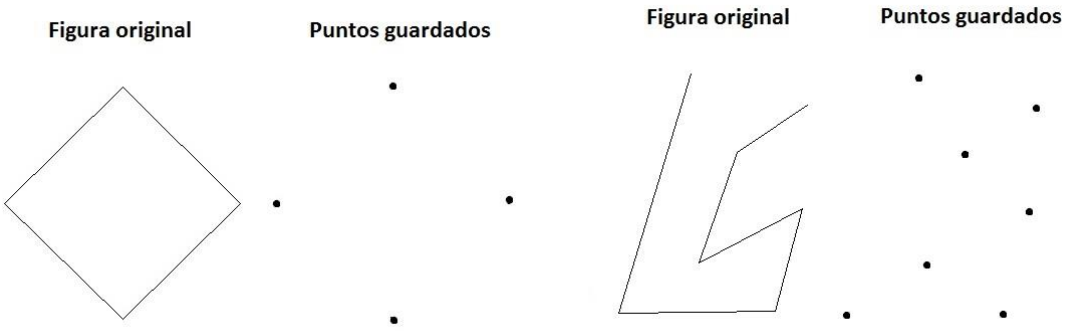


Figura 36. Original y puntos guardados para el cálculo de contornos de figuras formadas por líneas rectas

- **Filtro 3 Número de ángulos de cada tipo:**

Con estos primeros filtrados se ha conseguido tener un poco más acotado el número de objetos con posibilidades de ser una flecha, pero aún no es suficiente, puesto que puede haber gran cantidad de detecciones que superen el área mínima que se ha establecido en el primer filtrado y que estén compuestos por siete lados, pero que no sean flecha. Esto se puede observar en la *Figura 37,* en ella se muestran dos ejemplos en el que el algoritmo ha marcado detección de flecha (en morado) en una zona donde no hay flechas.

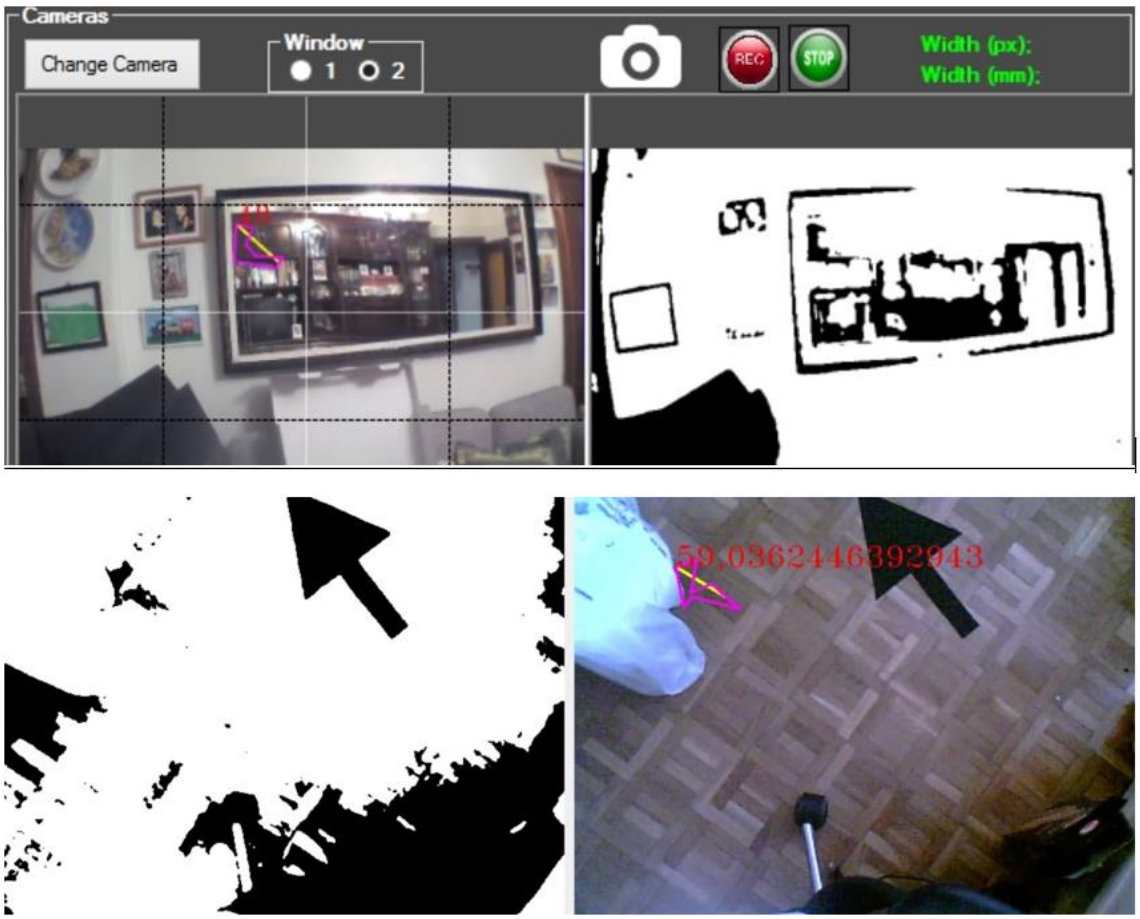


Figura 37. Falsa detección de flecha

Por otro lado, puede ser que el objeto detectado sí sea una flecha, pero que debido a una mala segmentación no se haya conseguido detectar su forma correctamente. Si esa detección se aceptara como válida acarrearía problemas en futuras fases del algoritmo, por lo que sería conveniente descartar también estas malas detecciones de flecha.

Por todo ello se necesita efectuar otro filtrado, en el que se analizarán los ángulos que forman los objetos.

Para el nuevo filtrado se van a establecer condiciones sobre el número y el tipo de ángulos que forman los objetos. Para ello el procedimiento que se sigue es el siguiente: Primero se calculan todos los ángulos y simultáneamente se van clasificando asumiendo que todos los objetos que se están analizando son flechas, por último se efectuará un filtrado para averiguar cuáles de ellos lo eran en realidad.

Cálculo: Se busca calcular los ángulos internos que forman los objetos, es decir, los que vienen dados por los lados del mismo. Por ello, antes de poder calcularlos deben encontrarse los lados del objeto.

Como se ha explicado, aquellas detecciones que hayan superado el segundo filtro serán figuras formadas por líneas rectas (ya sean polígonos o figuras abiertas *ver Figura 36*), de las cuales se tendrán almacenadas sus vértices en un array, que se denominará "Array de Vértices" **AV**. Los lados de la figura se pueden definir como segmentos que están delimitados por sus vértices, por lo tanto para encontrar los lados se va a partir de los vértices almacenados. Los lados se irán guardando en un vector que se llamará "Vector de Lados" **VL**.

Para calcular los lados se hace uso del método **".PolyLine"**. Este método recibe dos parámetros: El primero de ellos es el **AV**, el segundo es una variable booleana que se establece como verdadera. Lo que hace **".PolyLine"** es transformar la serie de vértices que son del tipo **"Point"** (punto) a una serie del tipo **"LineSegment2D"** (líneas). Con la booleana verdadera se indica que el último segmento de línea estará delimitado por el último y primer puntos del array, es decir por el último y primer vértice almacenados. Con esto se fuerza a que todas las figuras formadas por líneas rectas que habían superado el segundo filtro sean cerradas, por lo que a partir de este punto sólo se tendrán polígonos.

A continuación se explica procedimiento detallado que utiliza el método **".PolyLine"** para obtener la serie de lados. Será más fácil de comprender observando las *Figuras 38 y 39.* En la *Figura 38* se muestra de forma gráfica el procedimiento de obtención de lados, mientras que en la *Figura 39* se muestra lo mismo pero representando cómo quedarían almacenados en el **VL**:

Primero cabe destacar que los vértices de la figura están guardados en el array ocupando las posiciones de 0 – 6 (números de color naranja en las *Figuras 38 y 39*), y los lados calculados se van a almacenar en el **VL** ocupando también las posiciones de 0 – 6 (números de color azul y entre corchetes en las *Figuras 38 y 39*).

Se parte del **AV** (parte superior izquierda de la *Figura 38),* y se empieza calculando el primer lado, para ello toma como punto de inicio del segmento el vértice que ocupa la posición 0 del array, y como punto final el vértice almacenado en la siguiente posición, la 1 (parte superior derecha de la *Figura 38)*. El lado resultante se almacena en la

posición {0} del **VL** *ver Figura 39*. En el segundo lado esta vez se toma como punto de inicio el vértice que marca el final del primer lado, el número uno, y como punto final el vértice dos (parte media izquierda de la *Figura 38)*. Se continúa de la misma manera hasta que se ha llegado al último vértice almacenado. En este punto hay dos opciones: Si la variable que se transmite como parámetro a "**`.PolyLine`**" es falsa ya se tendrían calculados todos los lados, y el resultado sería el mostrado en la parte inferior izquierda de la *Figura 38,* por el contrario si fuera verdadera, se calcularía un lado más cuyos extremos serían el último vértice almacenado y el primero, ver parte inferior derecha de la *Figura 38*.

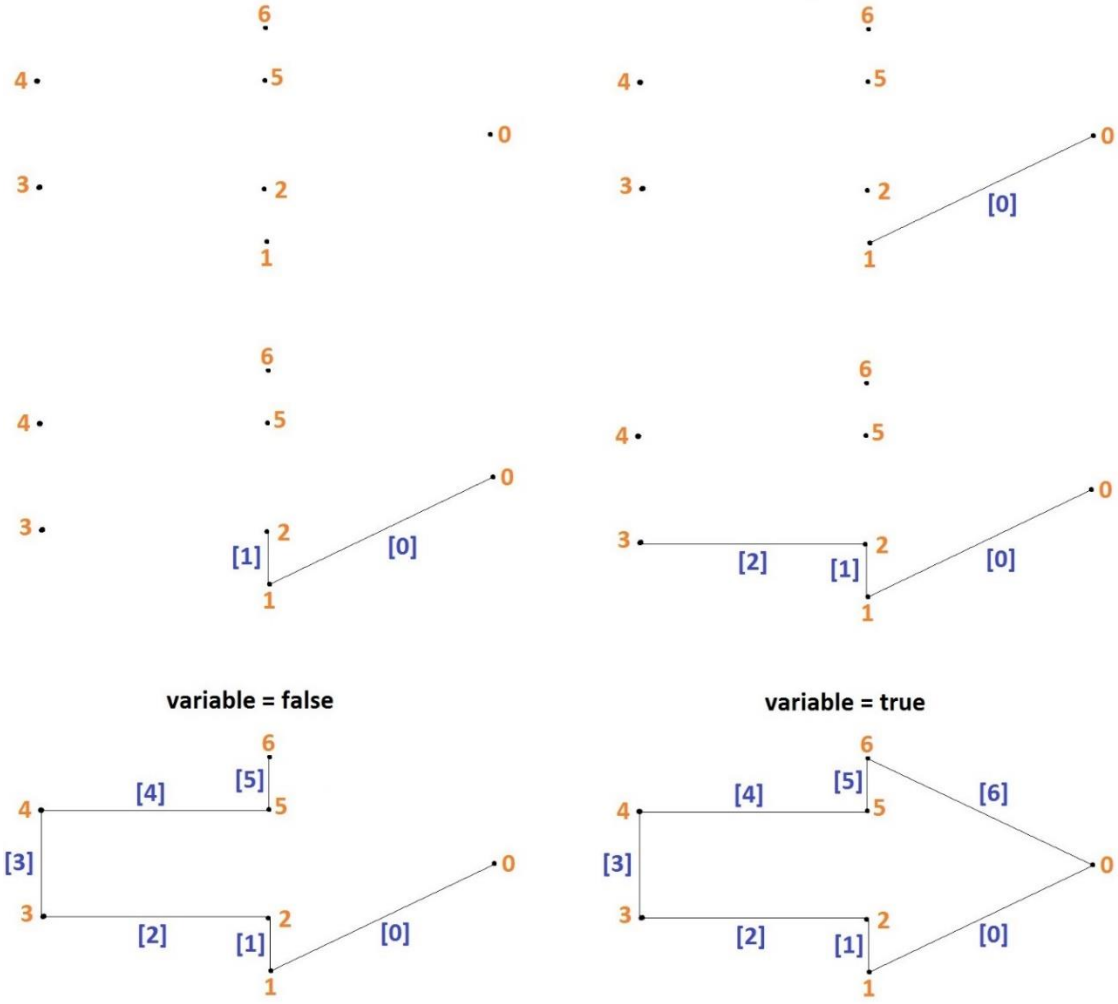


Figura 38. Pasos en la obtención de los lados de un polígono

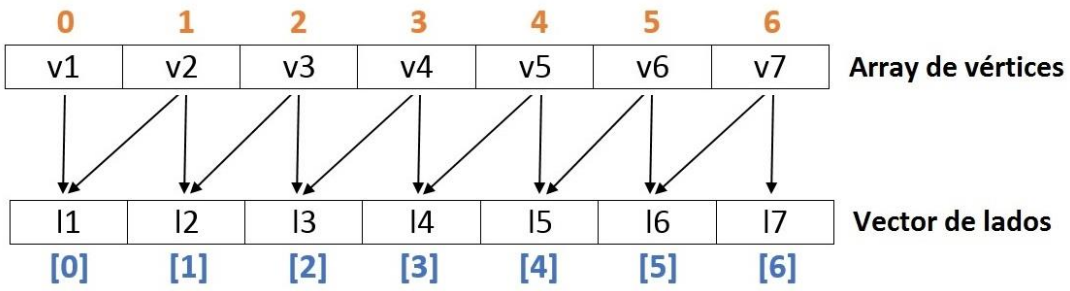


Figura 39. Representación de contenido, y posiciones del "array de vértices" y "vector de lados"

Una vez calculados los lados, se pueden calcular los ángulos internos del polígono gracias al método **“.GetExteriorAngleDegree()”** Este método calcula el ángulo exterior entre dos líneas del tipo “**LineSegment2D**”. El procedimiento para ir calculándolos es utilizar los lados en orden, tal y como están almacenadas en el vector. Primero se calculará el ángulo formado por el lado almacenado en la posición cero con el de la posición uno, después el ángulo formado por el almacenado en la posición uno con el de la dos y así sucesivamente.

Para obtener los ángulos interiores, que son los buscados, basta con restarle a 180º el ángulo exterior *ver Figura 40.*

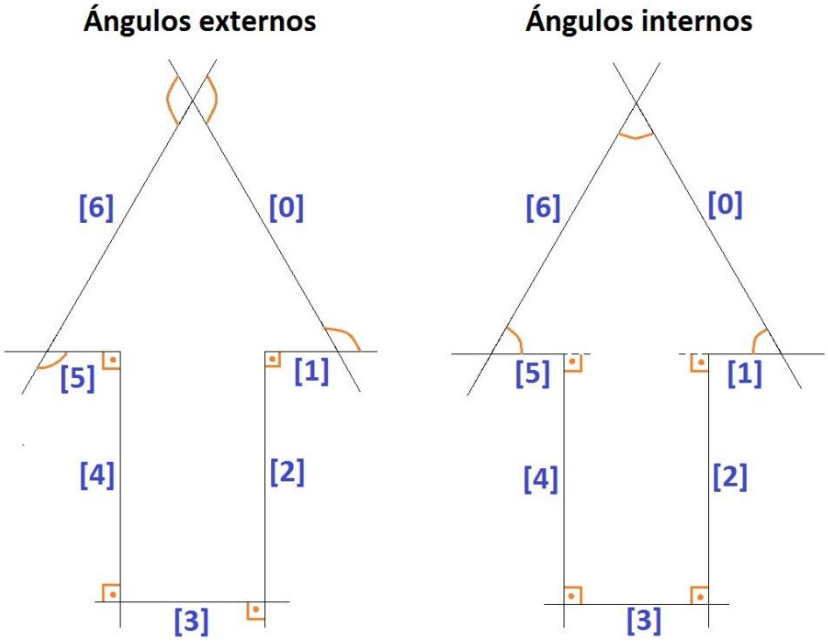


Figura 40. Ángulos externos e internos

Clasificación: A continuación se va a explicar el método seguido para clasificar los ángulos internos. Cabe destacar que ambos procesos se hacen de forma simultánea: primero se calcula un ángulo y acto seguido se clasifica, después se calcula el siguiente ángulo y se clasifica, y así sucesivamente. Para la clasificación se va a asumir que el objeto que se está analizando es una flecha, es decir, se van a ir clasificando los ángulos en base a características típicas de las flechas. Finalmente será tarea de la condición de filtrado eliminar aquellos objetos que no lo sean.

Tal como se explicó en el *apartado 4.4.3.1,* la flecha puede dividirse en dos zonas. En cada una de ellas los ángulos interiores presentan características comunes, por lo que para el filtrado posterior va a ser útil conocer a qué zona pertenece cada ángulo calculado.

En este punto se tiene un array en el que se han almacenado los vértices de la figura detectada **AV**, y un vector en el que se han guardado los lados de la misma **VL**. Para efectuar la clasificación se van a crear dos nuevos vectores “vector triángulo” **VT** en el que se van a guardar los tres ángulos correspondientes a la “zona de triángulo” **ZT**, y en otro vector “vector rectángulo” **VR** se guardarán todos los demás, que serán los correspondientes a la “zona rectángulo” **ZR**.

En la *Figura 41* los números naranjas representan la posición en la que están almacenados los vértices de la flecha en el **AV**, mientras que los números azules entre corchetes indican la posición en la que están almacenados los lados de la flecha en el **VL**. En la parte superior de la figura se muestra un ejemplo de los ángulos que deberían guardarse en el **VT**: serían los formados por los lados guardados en las posiciones {0} y {1}, {5} y {6} y {6} y {0}. Sin embargo, los vértices y lados no siempre se guardan en las mismas posiciones. Si se observa la parte inferior de la figura se podrá comprobar que esta vez los ángulos correspondientes a la zona de triángulo serán los formados por los lados que ocupan las posiciones {4} y {5}, {2} y {3} y {3} y {4}.

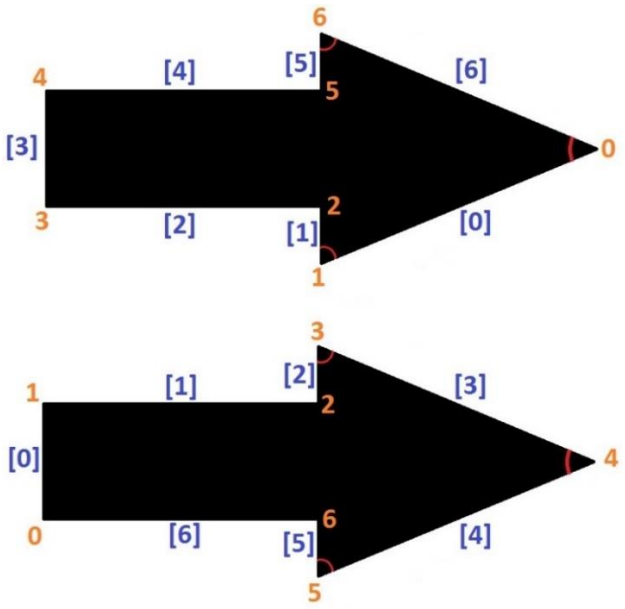


Figura 41. Vértices y lados de flecha ocupando diferentes posiciones de vector

Por lo tanto, para poder clasificar en el **VT** los ángulos marcados en rojo en la *Figura 41*, es necesario averiguar en qué posición se han almacenado los lados que los definen en el **VL**. Para ello, basta con averiguar en qué posición se ha guardado el "vértice de dirección" **VD**. Sabiendo este dato se puede conocer en qué posición están guardados todos los demás vértices, y por extensión todos los lados gracias a una propiedad que se explicará más adelante.

Llegados a este punto es necesario calcular el **VD** *ver Figura 42:*

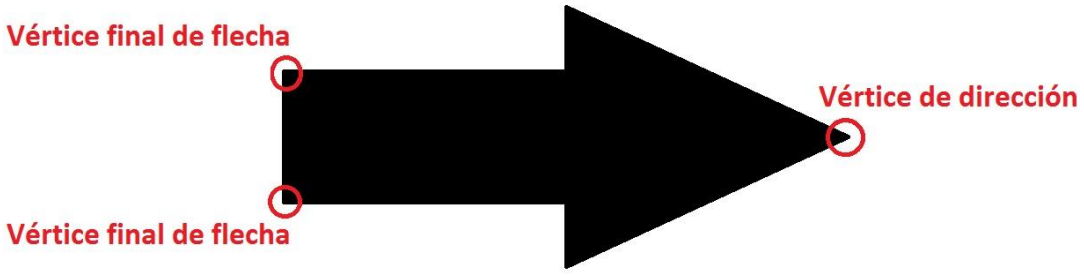


Figura 42. "Vértice de dirección" y "Vértices final de flecha"

De entre todos los vértices almacenados en el array, el **VD** y cualquiera de los dos "Vértices Final de Flecha" **Vff** cumplen la propiedad de que son los que están situados

a mayor distancia entre sí. Para encontrar sus posiciones, el método a seguir va a ser ir recorriendo el array, calculando para cada posición la distancia del vértice contenido en ella con respecto a todos los demás. Para ello lo que se hace es restar la componente "X" del vértice actual a la componente "X" del vértice siguiente, y lo mismo se hace con las componentes "Y" *ver Figura 43:*

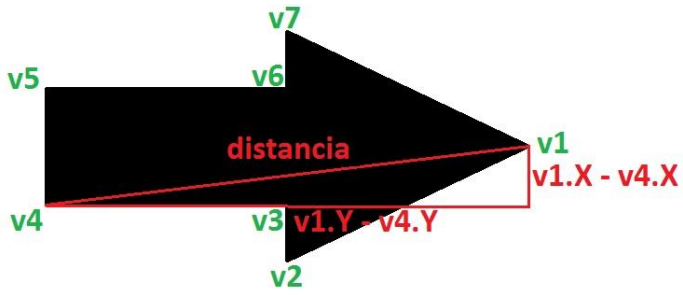


Figura 43. Ejemplo de cálculo de distancia entre dos vértices (v1 y v4)

El valor de distancia será el resultado de aplicar el teorema de Pitágoras *ver Ecuación 4.1*:

$$distancia = \sqrt{(vértice\ actual.X - vértice\ siguiente.X)^2 + (vértice\ actual.Y - vértice\ siguiente.Y)^2} \quad \textbf{(4.1)}$$

De esta operación se guardará en una variable la mayor distancia, y en otras dos variables las posiciones de los vértices que se encuentran a esa distancia. De tal manera que si analizando el resto del array se encuentra una distancia mayor a la guardada, se sobrescribirán las variables de distancia y posición. Una vez que se haya recorrido el array entero, se tendrán guardadas en esas variables la mayor distancia encontrada, y las posiciones de los vértices, que en este punto ya se corresponderán con el **VD** y uno de los dos **Vff**. Se indica que será uno de los dos porque ambos están situados a la misma distancia del **VD**, que sea uno o el otro dependerá de lo que se capta en cada momento por la cámara.

Obtenida la posición del **VD** ya se puede encontrar en qué posición del **VL** se almacenan los lados de la **ZT**, el método utilizado para ello se va a explicar a partir de dos ejemplos:

- *Ejemplo 1:*
  Como se observa en la *Figura 44*, el **VD** ocupa la posición 0 del array de vértices. Y los ángulos que deben almacenarse en el **VT** son los formados por los lados guardados en las posiciones {0} y {1}, {5} y {6}, {6} y {0} del **VL** marcados en rojo en la *Figura 44*.

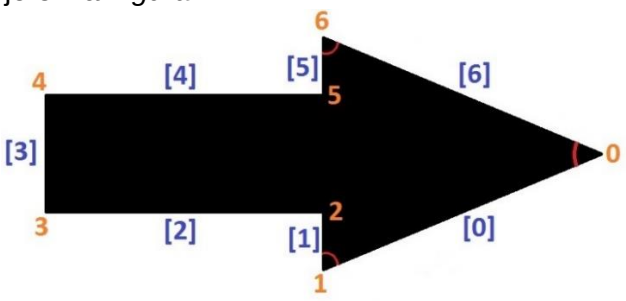


Figura 44. Posición de vértices y lados en Ejemplo 1

En la *Figura 45* se muestra la relación de ángulos internos (columna izquierda), los lados que los definen indicados por las posiciones reales que ocupan en el **VL** (columna de en medio) y los mismos lados pero esta vez con sus posiciones indicadas en función del dato conocido que es el de la "posición del Vértice de Dirección" **Pvd**.

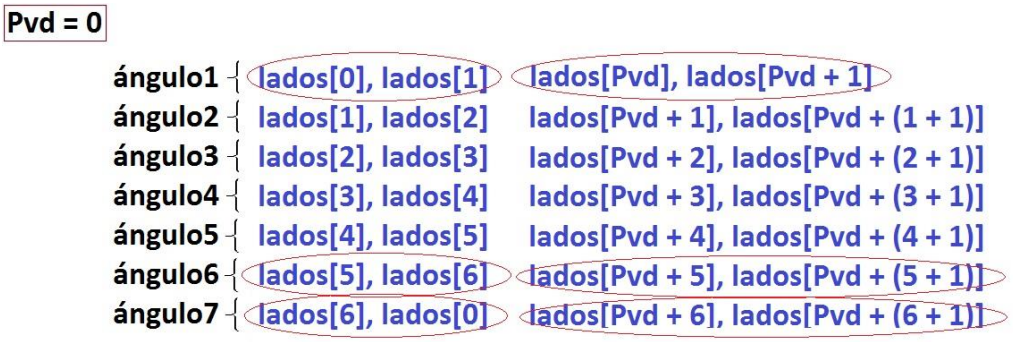


Figura 45. Ángulos internos y posiciones de los lados que los definen (reales y en función de Pvd)

Como se puede observar en la *Figura 45*, hay tres números clave que se le suman a la **Pvd** y que permiten encontrar a su vez la posición de los lados que se están buscando, los números clave son el 0, el 5 y el 6. *Ver Ecuaciones 4.2, 4.3 y 4.4:*

$$Pvd + 0 = 0 + 0 = 0 \rightarrow lados[Pvd + 0] = lados[\mathbf{0}] \quad \textbf{(4.2)}$$
$$Pvd + 5 = 0 + 5 = 5 \rightarrow lados[Pvd + 5] = lados[\mathbf{5}] \quad \textbf{(4.3)}$$
$$Pvd + 6 = 0 + 6 = 6 \rightarrow lados[Pvd + 6] = lados[\mathbf{6}] \quad \textbf{(4.4)}$$

Se recuerda que los ángulos que deben almacenarse en el **VT** en este ejemplo, son los formados por los lados guardados en las posiciones **{0}** y {1}, **{5}** y {6}, **{6}** y {0} del **VL**. Con las ecuaciones anteriores se han obtenido ya las primeras posiciones, marcadas en negrita. Para obtener las posiciones que faltan correspondientes al otro lado que define el ángulo, basta con sumarle una unidad a los números clave citados antes *ver Ecuaciones 4.5, 4.6 y 4.7*:

$$Pvd + (0 + 1) = 0 + (0 + 1) = 1 \rightarrow lados[Pvd + (0 + 1)] = lados[\mathbf{1}] \quad \textbf{(4.5)}$$
$$Pvd + (5 + 1) = 0 + (5 + 1) = 6 \rightarrow lados[Pvd + (5 + 1)] = lados[\mathbf{6}] \quad \textbf{(4.6)}$$
$$Pvd + (6 + 1) = 0 + (6 + 1) = 7 \rightarrow lados[Pvd + (6 + 1)] = lados[7] \quad \textbf{(4.7)}$$

Gracias a estas ecuaciones ya se han obtenido las posiciones **{0}** y **{1}**, **{5}**, **{6}** y **{6}** y {0} del **VL**, marcadas en negrita. Si se analiza el resultado de la última ecuación a priori podría parecer que es erróneo, puesto que las posiciones del **AV** se enumeran del 0 – 6, y no hay una séptima posición. Esa ecuación debería dar como resultado 0. En realidad no es errónea, basta con añadirle a las ecuaciones un término más para que den los resultados correctos en todos los casos. Ese término es añadirle el operador resto "%": El resto de la división del resultado de las ecuaciones anteriores entre el total de lados almacenados en el vector, siete, será el que marque las posiciones buscadas. A continuación

se repiten las ecuaciones ordenadas por pares de lados y con el operador resto *ver Ecuaciones 4.8, 4.9, 4.10, 4.11, 4.12 y 4.13*:

$$Pvd + 0 = (0 + 0)\ \%\ 7 = 0 \rightarrow lados[(Pvd + 0)\ \%\ 7] = lados[\mathbf{0}] \quad \textbf{(4.8)}$$
$$Pvd + (0 + 1) = (0 + 1)\ \%\ 7 = 1 \rightarrow lados\big[\big(Pvd + (0 + 1)\big)\ \%\ 7\big] = lados[\mathbf{1}] \quad \textbf{(4.9)}$$
$$Pvd + 5 = (0 + 5)\ \%\ 7 = 5 \rightarrow lados[(Pvd + 5)\ \%\ 7] = lados[\mathbf{5}] \quad \textbf{(4.10)}$$
$$Pvd + (5 + 1) = (0 + 6)\ \%\ 7 = 6 \rightarrow lados\big[\big(Pvd + (5 + 1)\big)\%\ 7\big] = lados[\mathbf{6}] \quad \textbf{(4.11)}$$
$$Pvd + 6 = (0 + 6)\ \%\ 7 = 6 \rightarrow lados[(Pvd + 6)\%\ 7] = lados[\mathbf{6}] \quad \textbf{(4.12)}$$
$$Pvd + (6 + 1) = (0 + 7)\%\ 7 = 0 \rightarrow lados\big[\big(Pvd + (6 + 1)\big)\%\ 7\big] = lados[\mathbf{0}] \quad \textbf{(4.13)}$$

Ahora ya sí se han obtenido las posiciones de todos los lados buscados **{0}** y **{1}**, **{5}** y **{6}**, **{6}** y **{0}**.
En el siguiente ejemplo se va a demostrar que estas ecuaciones también sirven para un número de posición de **VD** diferente.

- *Ejemplo 2:*
  En este caso, como se observa en la *Figura 46*, el **VD** ocupa la posición 4 del **AV**. Y los ángulos que deben almacenarse en el **VT** son los formados por los lados guardados en las posiciones {4} y {5}, {2} y {3}, {3} y {4} del **VL**, marcados en rojo en la *Figura 46*.

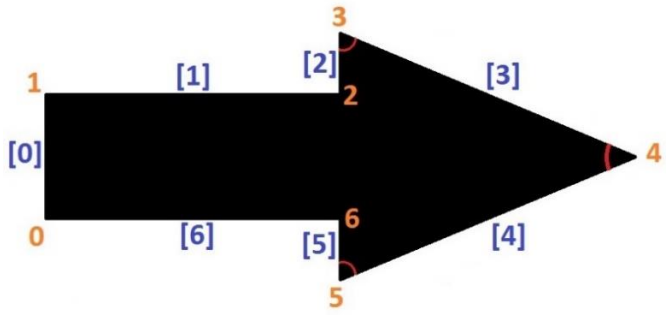


Figura 46. Posición de vértices y lados en Ejemplo 2

Las ecuaciones en este caso serían, *Ver Ecuaciones 4.14, 4.15, 4.16, 4.17, 4.18 y 4.19*:

$$Pvd + 0 = (4 + 0)\ \%\ 7 = 4 \rightarrow lados[(Pvd + 0)\ \%\ 7] = lados[\mathbf{4}] \quad \textbf{(4.14)}$$
$$Pvd + (0 + 1) = (4 + 1)\ \%\ 7 = 5 \rightarrow lados\big[\big(Pvd + (0 + 1)\big)\%\ 7\big] = lados[\mathbf{5}] \quad \textbf{(4.15)}$$
$$Pvd + 5 = (4 + 5)\ \%\ 7 = 2 \rightarrow lados[(Pvd + 5)\ \%\ 7] = lados[\mathbf{2}] \quad \textbf{(4.16)}$$
$$Pvd + (5 + 1) = (4 + 6)\ \%\ 3 = \rightarrow lados\big[\big(Pvd + (5 + 1)\big)\ \%\ 7\big] = lados[\mathbf{3}] \quad \textbf{(4.17)}$$
$$Pvd + 6 = (4 + 6)\ \%\ 7 = 3 \rightarrow lados[(Pvd + 6)\ \%\ 7] = lados[\mathbf{3}] \quad \textbf{(4.18)}$$
$$Pvd + (6 + 1) = (4 + 7)\%\ 7 = 4 \rightarrow lados\big[\big(Pvd + (6 + 1)\big)\ \%\ 7\big] = lados[\mathbf{4}] \quad \textbf{(4.19)}$$

Con ellas se han obtenido las posiciones de los lados buscados **{4}** y **{5}**, **{2}** y **{3}**, **{3}** y **{4}**.

Las ecuaciones expuestas para estos dos ejemplos son válidas para todos los valores de posición que puede tener el **VD**. Gracias a estas ecuaciones ya se puede saber en qué posiciones del **VL** se encuentran almacenados los lados de cada zona, por lo que se puede iniciar la clasificación.

Para la correcta comprensión del método que se va a seguir en la clasificación de los ángulos es necesario retomar un par de aspectos: La explicación de cómo se calculan los ángulos internos explicada en el "Filtro 3 Número de ángulos de cada tipo" y también que el cálculo de ángulos y la clasificación se hacen de forma simultánea (comentado al inicio de este apartado).

A la hora de calcular los ángulos internos se va a utilizar un bucle **"`for`"**. Se establecerá que el bucle se repita siete veces gracias a una variable entera **"`i`"** que se irá incrementando en cada repetición. Dentro de este bucle se efectúa el cálculo de ángulos internos sirviéndose del método **"`.GetExteriorAngleDegree()`"** en el que se acoplan las ecuaciones descritas anteriormente de la siguiente manera *ver Ecuación 4.20*:

$$angle = 180 - (lados\big[\big(Pvd + (i+1)\big)\ \%\ 7\big].GetExteriorAngleDegree(lados[(Pvd + i)\ \%\ 7]); \quad \textbf{(4.20)}$$

Cuando esta variable tome los valores que se han establecido en las ecuaciones como números clave "0, 5 y 6" en la explicación del Ejemplo 1, los ángulos calculados con ese valor de **"`i`"** se almacenarán en el **VT**, cuando tome el resto de valores se almacenarán en el **VR**.

La clasificación tiene una vuelta de tuerca más. Como se dijo en éste apartado a la hora de obtener la **Pvd**, se obtiene también la posición de otro, el del **Vff**. Gracias al filtrado se va a poder conocer cuál de las dos posiciones obtenidas corresponde al **VD** y cuál al **Vff**. Como en este punto el algoritmo no es capaz de discernirlo se van a calcular dos ángulos, el primero de ellos asumirá como **Pvd** el dato de una de las posiciones obtenidas, y el segundo asumirá como **Pvd** el dato de la otra posición de la que se dispone. Los ángulos calculados con la primera posición se guardarán en el "vector triángulo 1" **VT1**, mientras que los ángulos calculados con la segunda posición se guardarán en el "vector triángulo 2" **VT2**. Con esto se va a obtener dos vectores triángulo, uno con los tres ángulos que realmente corresponden a la **ZT** y otro que habrá que descartar porque contendrá tres ángulos correspondientes a la **ZR**. Del mismo modo, se obtendrán dos vectores rectángulo, uno con los cuatro ángulos correspondientes a la **ZR**, y otro que habrá que descartar porque contendrá ángulos correspondientes a ambas zonas.

Filtrado: La condición de filtrado va a tener doble propósito, descartar detecciones que no sean una flecha, y además discernir los siguientes aspectos:

- Cuál de las dos posiciones que se obtuvieron corresponden al **VD** y cuál al **Vff** (lo cual será útil para posteriores fases del algoritmo).
- Cuál de los dos **VT** es el que contiene los ángulos correctos, y cuál de ellos hay que descartar.
- Cuál de los dos **VR** es el que contiene los ángulos correctos, y cuál de ellos hay que descartar.

La condición de filtrado que se va a establecer tiene que ver con el número y tipo de ángulos internos de cada zona. La **ZT** deberá tener una mayoría de ángulos agudos, mientras que la **ZR** deberá estar formada por ángulos rectos *ver apartado 4.4.3.1.* Por ello, antes de filtrar se recorrerán los dos **VT** contando el número de ángulos agudos que contiene cada uno, y de la misma manera se recorrerán los dos **VR** contando el número de ángulos rectos que contiene cada uno de ellos. Debido a que la segmentación realizada no sea del todo buena, o dependiendo del ángulo de la

cámara, puede darse el caso de que cuando se calcula el tipo de ángulo los rectos no se correspondan exactamente con 90º, ni los agudos con menores de 90º, sino que estén dentro de un intervalo. Para averiguar cuál es el intervalo más adecuado se han realizado varias pruebas, obteniendo finalmente los siguientes:

- Se considerará ángulo recto a aquellos ángulos cuyo valor se encuentre dentro del intervalo: 81º < ángulo < 121º
- Se considerará ángulo agudo a aquellos ángulos cuyo valor se encuentre dentro del intervalo 0º < ángulo < 81º

Con esta información se puede establecer la condición de filtro:

Idealmente las flechas podrían estar formadas por las siguientes combinaciones de ángulos (*ver apartado 4.4.3.1* para una mejor comprensión):

- Cuatro ángulos rectos en la **ZR** y tres agudos en la **ZT**.
- Cuatro ángulos rectos en la **ZR**, y dos agudos y un recto en la **ZT**.
- Cuatro ángulos rectos en la **ZR**, y dos agudos y un obtuso en la **ZT**.
- Tres ángulos rectos y dos agudos en la **ZR**, y tres agudos en la **ZT**.

Sin embargo debido a que la segmentación realizada puede no ser del todo buena, podría darse el caso de que algún ángulo que debería ser recto se detecte como obtuso o agudo etc, *ver Figura 47*. Es por eso que no se va a imponer una condición de filtro tan restrictiva.

Figura 47. Ángulo recto de flecha detectado como agudo

Por todo ello la condición de filtro que se va a establecer es la siguiente: Para que una detección sea considerada como flecha su **VT** debe contener un mínimo de dos ángulos agudos y un máximo de tres, mientras que su **VR** debe contener un mínimo de dos ángulos rectos y un máximo de cuatro. Como se ha comentado, se tiene un **VT1** y **VR1** cuyos ángulos han sido calculados con una de las dos posiciones obtenidas (la del **VD** o la del **Vff**), y otro **VT2** y **VR2** cuyos ángulos han sido calculados con la otra posición. Si ninguno de los dos pares de vectores supera el filtro significará que la detección no era una flecha. Si lo es, uno de los dos pares de vectores lo superará, y serán aquellos cuyos ángulos hayan sido calculados sirviéndose de la posición del **VD**.

Gracias a este filtro se ha conseguido descartar detecciones que no eran flechas, tener clasificados los ángulos internos por zonas, y conocer la posición del **VD** lo cual será de gran utilidad en futuras fases del algoritmo.

- **Filtro 4 Paralelismo:**

Tras tres filtros las posibilidades de que una detección no se corresponda con una flecha son mucho más remotas, pero aún son posibles. Además sigue existiendo el riesgo de que debido a una mala segmentación no se haya captado correctamente la forma de la flecha, y si se aceptara como válida acarrearía problemas en siguientes pasos del algoritmo. Para descartar las falsas o malas detecciones de flecha, se va a establecer una última condición de filtrado: que la forma de flecha contenga cuatro líneas paralelas.

El método seguido para saber si las líneas son paralelas es el siguiente:

En la forma de flecha las líneas paralelas aparecen en la zona rectángulo *ver apartado 4.4.3.1* por lo que la búsqueda del paralelismo se va a efectuar en esa zona. El primer paso es calcular las distancias marcadas en rojo en la *Figura 48.*

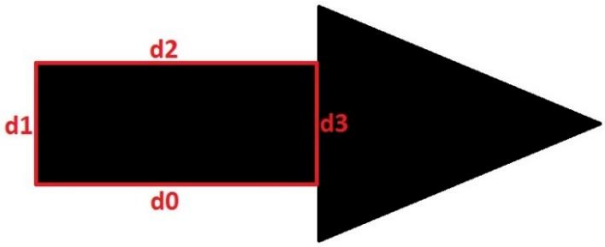


Figura 48. Distancias a calcular en la búsqueda del paralelismo

Una vez calculadas las distancias se va a establecer que si d1 y d3 son iguales, los lados l1 y l2 serán paralelos, mientras que si son diferentes no lo serán *ver Figura 49.* En el ejemplo concreto de la *Figura 49*, aunque los lados l1 y l2 no guarden paralelismo la forma sigue recordando a una flecha. Sin embargo, esa detección no se va a considerar por buena porque como se verá en siguientes fases del algoritmo, supondría un error en el cálculo de su dirección, por lo que el guiado del dron se efectuaría de forma incorrecta.

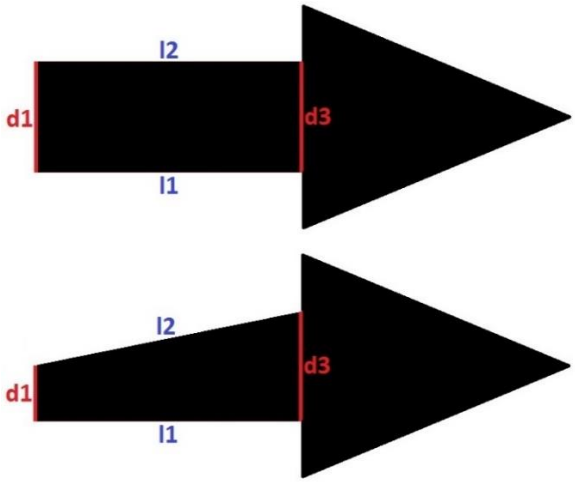


Figura 49. Ejemplos de la repercusión que tiene en l1 y l2 distancias d1 y d3 iguales o diferentes

De la misma manera se va a establecer que si las distancias d0 y d2 son iguales, los lados l2 y el lado auxiliar l3 serán paralelos, mientras que si son diferentes no lo serán

ver Figura 50. En el ejemplo concreto de la *Figura 50* se ve que con la condición establecida se consigue eliminar una detección que habría pasado los filtros anteriores, que se parece a una flecha pero que en realidad no lo es, y si se considerara como tal repercutiría en errores graves en siguientes fases del algoritmo.

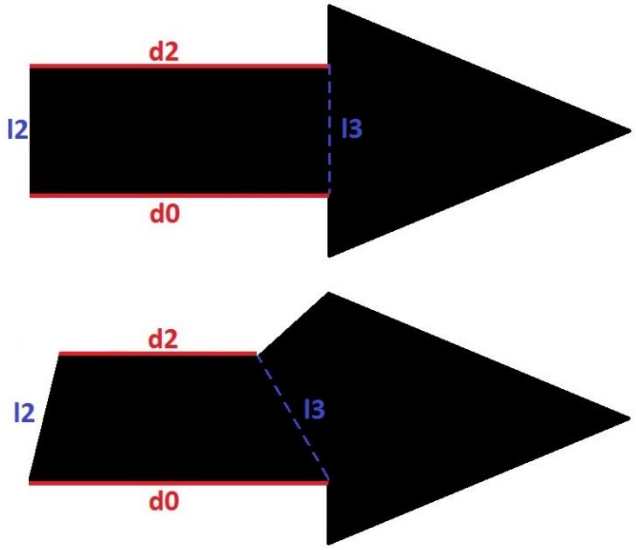


Figura 50. Ejemplos de la repercusión que tiene en l2 y l3 distancias d0 y d2 iguales o diferentes

Una vez que se ha explicado la filosofía que se sigue en este filtrado, se va a exponer cómo se traduce en el algoritmo:

Lo que se va a hacer es acceder a las posiciones en las que están guardados los vértices de la zona rectángulo en el **AV** sirviéndose del dato de la posición del **VD** que es conocido. De esta manera se va a ir calculando la distancia del vértice actual con respecto al siguiente almacenado en el array, hasta que se llegue al último vértice de la **ZR** en el que se calculará la distancia entre ese vértice y el primero de la **ZR**. El método matemático para calcular las distancias que en las *Figuras 49 y 50* vienen representadas por la nomenclatura d0, d1, d2 y d3 es el mismo que el explicado en el anteriormente en este apartado, se restan las componentes “X” e “Y” del vértice actual y el siguiente, y posteriormente se aplica el teorema de Pitágoras.

Si la detección cumple las condiciones de los cuatro filtrados expuestos se asumirá que es una flecha y que además la segmentación realizada es lo suficientemente buena como para no acarrear problemas en futuras fases del algoritmo, mientras que si no ha conseguido superar todos los filtros significará o que no era una flecha o que no se ha conseguido detectar bien su forma debido a mala segmentación u otras causas y se procederá a descartarla.

En el algoritmo se tiene una variable booleana que terminará siendo verdadera si la detección ha pasado correctamente todos los filtrados, y falsa si no los ha superado. Si la variable es verdadera se puede proceder al análisis de la detección.

4.4.4 Guiado

El objetivo del trabajo es que un dron sea capaz de seguir de forma autónoma y a tiempo real un camino formado por flechas. Para ello, una vez que se ha conseguido

efectuar la correcta detección y reconocimiento de las flechas, se puede comenzar a analizarlas con el fin de extraer la información necesaria para su guiado.

La información que se necesita es aquella que permita al dron girar en el sentido correcto hasta alinearse con la flecha una vez que la ha detectado. Hecho esto, deberá seguir volando en el sentido y dirección que marque la última flecha que detectó, ya que será en esa dirección donde encuentre la siguiente. Esto es lo que se conoce como guiado. Por tanto la información que es necesario extraer es la dirección y sentido de cada flecha.

En este apartado se va a comenzar explicando el procedimiento para extraer la dirección y sentido de las flechas, para después exponer cómo se traduce esa información a un ángulo que permita que el dron gire hasta alinearse con la flecha. Después se van a detallar los diferentes arreglos que se han programado para hacer el guiado lo más robusto posible. Finalmente se expondrá el método seguido para transmitir toda la información que se ha recopilado durante las diferentes fases del algoritmo al controlador del dron para que efectúe los movimientos correspondientes.

4.4.4.1 Dirección y sentido

La dirección de la flecha va a estar definida por una línea que vaya desde el **VD** hasta el punto medio de los **Vff**, en la *Figura 51* se corresponde con la línea amarilla.

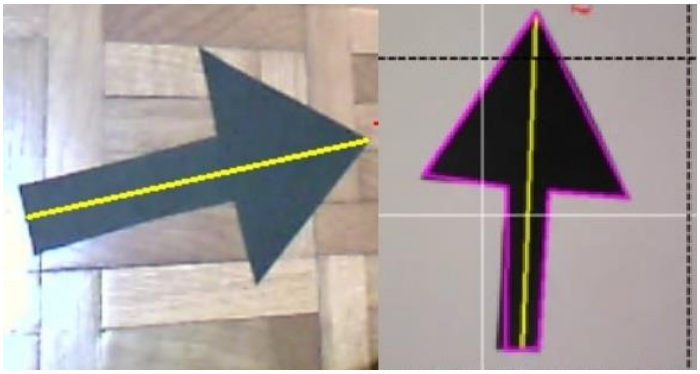


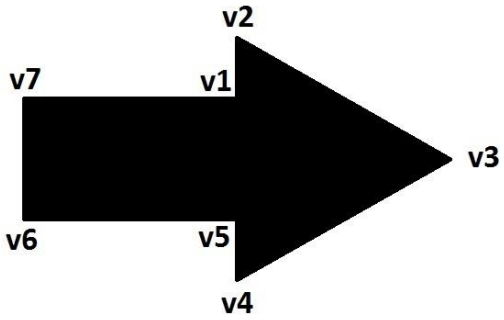
Figura 51. Línea de dirección de dos flechas

El modo de ejecutarlo en el algoritmo se va a explicar siguiendo la *Figura 52:*

En este punto el único dato de posición conocido es el del **VD**. A partir de ella se va a acceder a las "Posiciones de los Vértices final de flecha", denominados **Pff** y **Pff´** en la *Figura 52*. Para ello basta con darse cuenta que **Pff** será igual a la posición del vértice de dirección **Pvd** más tres, y **Pff´** será igual a **Pvd** más cuatro. Para que estas afirmaciones se cumplan en todas las posiciones que puede tomar el **VD** debe utilizarse el operador resto % entre el total de vértices contenidos en el array. En el caso concreto de la *Figura 52* estas operaciones quedarían como en las *Ecuaciones 4.21 y 4.22*:

$$Pff = (Pvd + 3)\ \%\ 7 = (2 + 3)\%\ 7 = 5 \qquad \textbf{(4.21)}$$

$$Pff\,' = (Pvd + 4)\%\ 7 = (2 + 4)\%\ 7 = 6 \qquad \textbf{(4.22)}$$



v1	v2	v3	v4	v5	v6	v7
0	1	2 = Pvd	3	4	5 = Pff	6 = Pff´

Figura 52. Ejemplo de cómo quedan almacenados en el array el "vértice de dirección" y los "vértices final de flecha"

Una vez que se ha conseguido acceder a **Pff** y **Pff´**, el siguiente paso es calcular el punto medio entre ambos. La coordenada "X" del punto medio **PM** estará definida por la suma de las coordenadas "X" de **Pff** y **Pff´** entre dos, del mismo modo la coordenada "Y" de **PM** estará definida por la suma de las coordenadas "Y" de **Pff** y **Pff´** entre dos, *ver Ecuaciones 4.23 y 4.24:*

$$PM.X = (Pff.X + Pff'.X)/2 \qquad \textbf{(4.23)}$$

$$PM.Y = (Pff.X + Pff'.X)/2 \qquad \textbf{(4.24)}$$

Con esto ya se tienen los dos puntos que definen la línea de dirección: El **VD** y el **PM**. Por lo que sólo queda crear la línea de dirección que será el segmento de línea definido por estos dos puntos. La línea de dirección será del tipo "**`LineSegment2D`**" que necesita como inputs los puntos que la definen, siendo crucial el orden al introducirlos ya que esto marcará el sentido de la flecha. El sentido de la flecha lo marca su propia forma (hacia donde esté apuntando en cada momento) y está definido por el **VD**. Es por ello que éste será el primer punto a introducir como input para crear la línea de dirección, mientras que el punto medio será el segundo *ver Ecuación 4.25:*

$$LineSegment2D\ DirectionLine = new\ LineSegment2D(VD, PM); \qquad \textbf{(4.25)}$$

Ahora que la dirección y el sentido de las flechas son conocidos, puede iniciarse la fase de extracción de información para el guiado del dron.

4.4.4.2 Cálculo de ángulos para el guiado del dron

Antes de explicar el procedimiento de cómo se transforma la información de dirección y sentido de una flecha a un ángulo que permita que el dron gire hasta alinearse con la misma conviene conocer los ángulos de navegación que posee el dron.

Los diferentes movimientos que el dron es capaz de efectuar durante su navegación vienen dados por los siguientes ángulos (ángulos de Euler) *ver Figura 53*:

Figura 53. Ángulos de navegación del dron

Como se observa en la figura el ángulo de cabeceo (Pitch) representa una inclinación de la cabeza del dron, el ángulo de guiñada (Yaw) representa una rotación respecto a un eje perpendicular al dron y el ángulo de alabeo (Roll) representa una rotación con respecto a un eje cabeza-cola del dron.

Al jugar con estos ángulos se consiguen los diferentes movimientos del dron. Variar el ángulo Roll permite obtener movimiento de derecha a izquierda, con Yaw se consigue que el dron rote sobre sí mismo, y al modificar Pitch se obtienen movimientos hacia delante y hacia atrás. No obstante el dron tiene un grado de libertad más, es el denominado Gaz, que le permite hacer movimientos de subida y de bajada.

Para conseguir variar estos ángulos se juega con el movimiento de las hélices y el par que producen. En la *Figura 54* se puede observar el sentido de giro de las hélices y cómo varían sus velocidades angulares en cada caso para conseguir variar los ángulos de Euler y para obtener el movimiento arriba y abajo correspondiente a Gaz. En la *Figura 54* Ω representa la velocidad angular de las hélices y ΔA, ΔB también son velocidades angulares positivas.

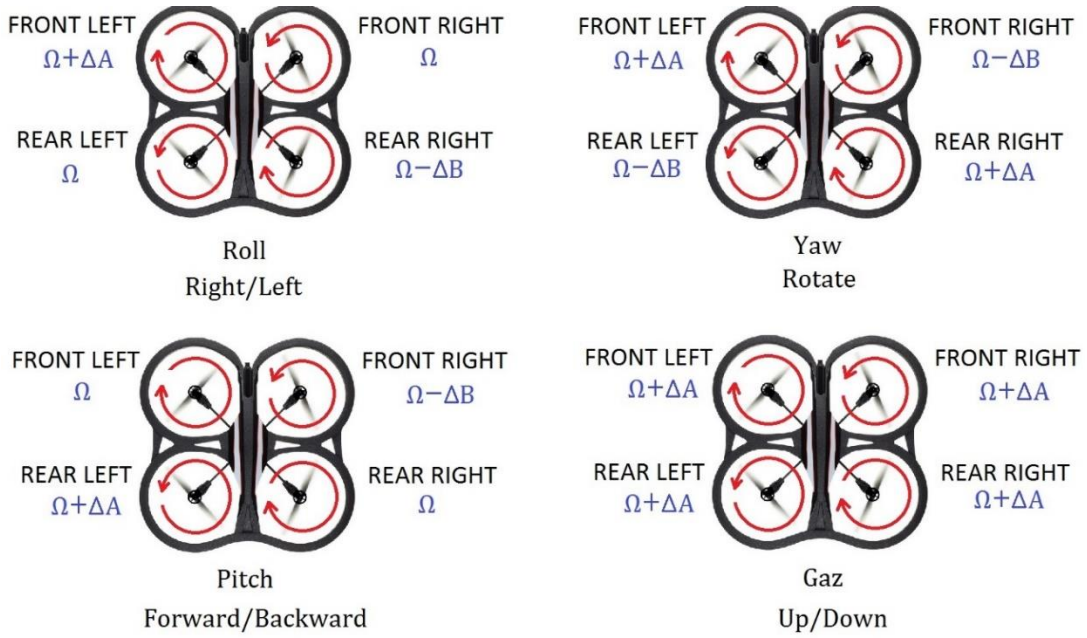


Figura 54. Representación del sentido de giro de las hélices y velocidades angulares para obtener cada uno de los movimientos

Como se verá más adelante para el guiado del dron por el camino de flechas, van a tener especial importancia los ángulos “Pitch” y “Yaw”, de este último se adjunta su referencia en la *Figura 55.*

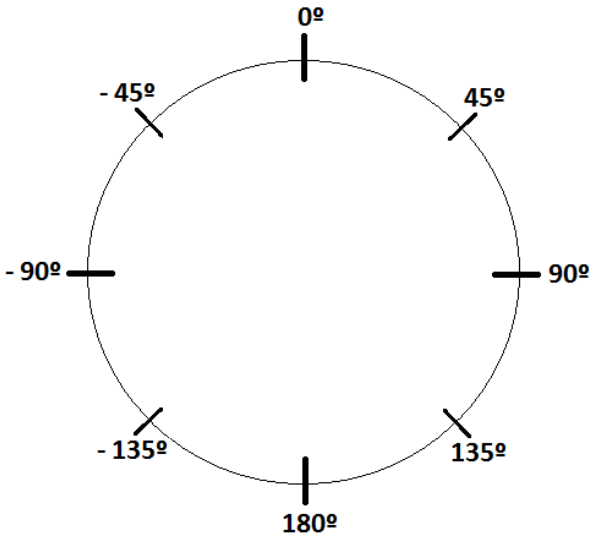


Figura 55. Referencia de Yaw

En el caso concreto de este trabajo se necesita que al encontrarse el dron con una flecha en una determinada dirección, gire un cierto ángulo hasta que quede alineado con la misma, es decir hasta que ambos estén en la misma dirección *ver Figura 56:*

Figura 56. Ejemplo del giro que efectúa el dron al encontrarse con una flecha

El ángulo que el dron debe girar hasta alinear su dirección con la que marca la flecha es el denominado Yaw, o ángulo de dirección. Es por ello que basándose en la referencia de Yaw que posee el dron mostrada en la *Figura 55*, ahora se puede ilustrar el ejemplo anterior con el ángulo que debería estar marcando la flecha para que ejecute el giro correctamente *ver Figura 57:*

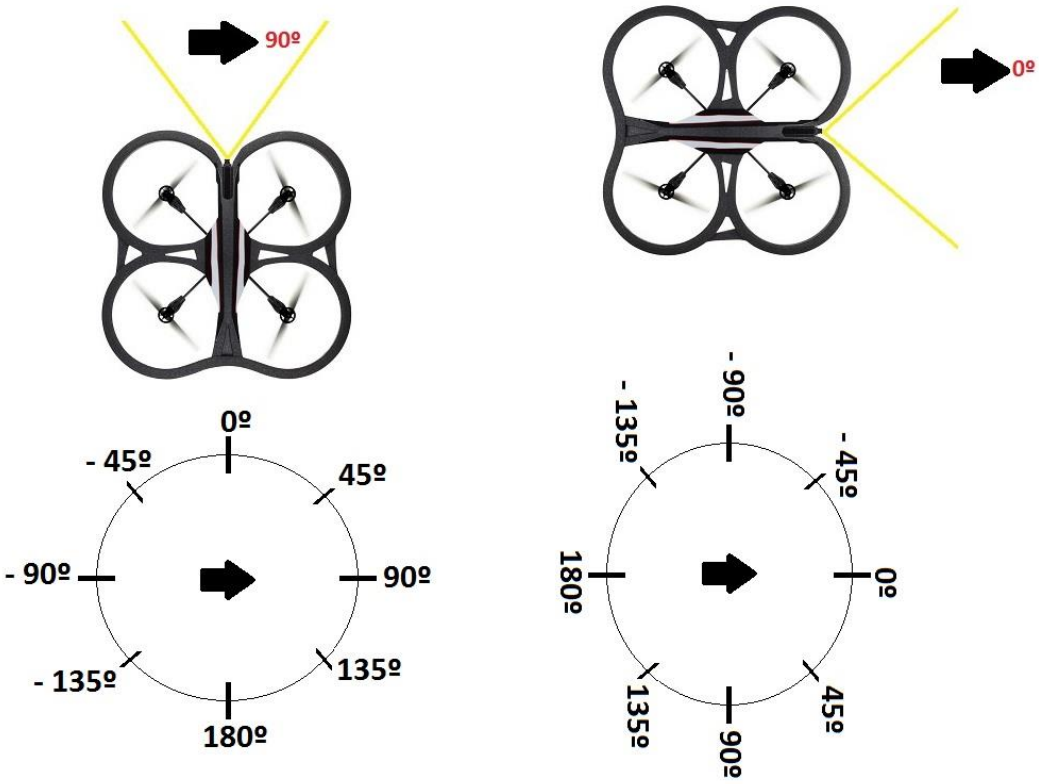


Figura 57. Ejemplo de giro del dron ilustrado con la referencia de los ángulos de Yaw

Como se puede observar en la *Figura 57* la referencia va unida al dron, y si éste gira la referencia gira con él. Para simular esto en el algoritmo se va a crear una línea a la que se denominará "línea de referencia de Yaw" **LRY** en el extremo derecho de la imagen que capta la cámara, que representará la línea imaginaria que une los ángulos 0º y 180º de la referencia de Yaw. De esta manera cuando el dron se mueva o gire, obviamente la imagen que capta la cámara girará con él, y con ella la referencia de Yaw. Esta línea está representada en color azul en la *Figura 58.*

Para obtener el ángulo de Yaw basta con calcular el ángulo que forma la línea de dirección de la flecha representada en amarillo en la *Figura 58*, con la **LRY** representada en azul. Para ello se utiliza como se ha visto en apartados anteriores, el método **"`GetExteriorAngleDegree();`"**

El proceso se va a comprender mejor con el ejemplo de la *Figura 58.* El primer paso consiste en la detección y reconocimiento de la flecha, acto seguido el algoritmo calculará el ángulo que forma la línea de dirección de la flecha con la **LRY**, observando que en el ejemplo ese cálculo da como resultado 90º. Una vez calculado Yaw, el dron comenzaría a girar y con él la referencia representada por la línea azul, por tanto el ángulo que forma la dirección de la flecha con la línea de referencia se hace cada vez más pequeño hasta llegar a 0º. En ese punto el giro se detiene porque el dron ha conseguido alinear su dirección con la de la flecha.

Cabe resaltar por tanto, que el ángulo de Yaw no es sólo el primero que se calcula al detectar la flecha (en el ejemplo de la figura 90º) sino que a cada frame debido al giro del dron, ese ángulo Yaw va cambiando haciéndose más pequeño hasta llegar a 0º.

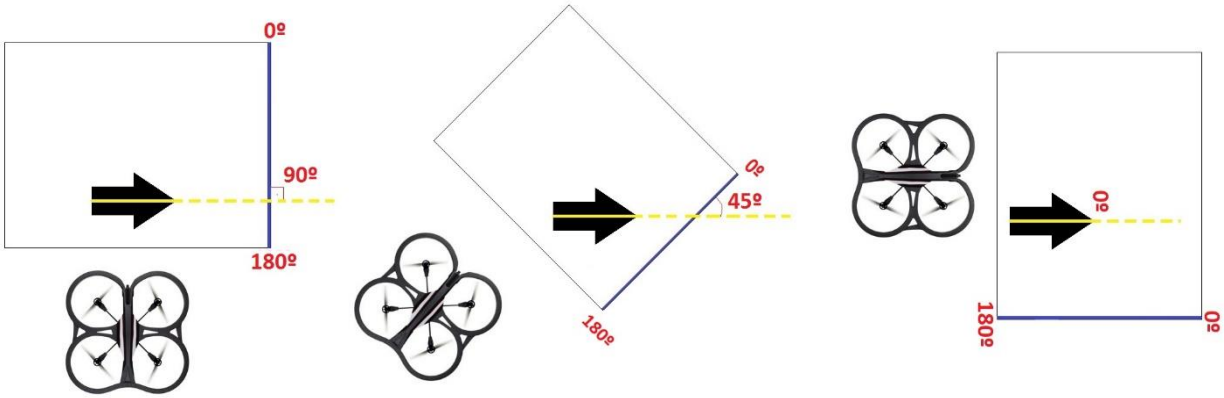


Figura 58. Ejemplo 1 del proceso detallado de giro: Frame captado por la cámara y posición del dron

En el ejemplo de la *Figura 59* se observa que el dron se encuentra con una flecha justo en sentido contrario que en el ejemplo anterior, sin embargo, el ángulo que forma la línea de dirección con respecto a la de referencia de Yaw es el mismo, 90º, cuando debería ser de – 90º. Es por ello que en el ejemplo se observa que el dron comienza a girar en la dirección equivocada.

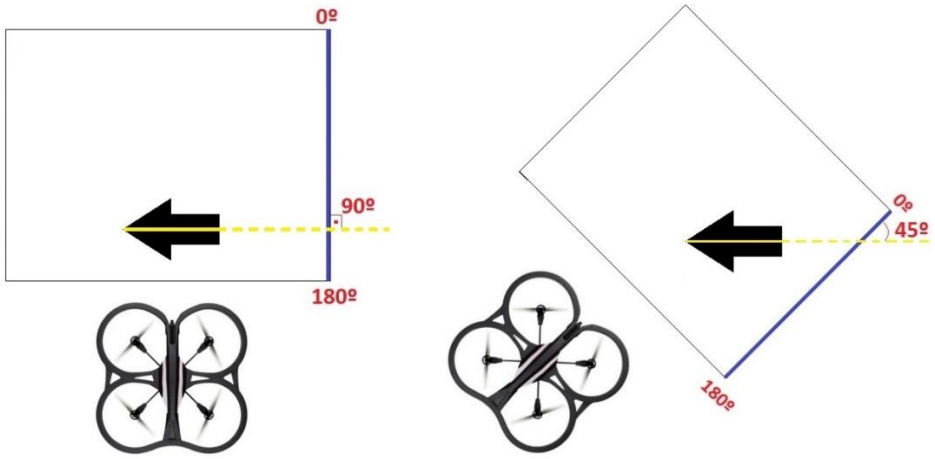


Figura 59. Ejemplo 2 del proceso detallado de giro: Frame captado por la cámara y posición del dron

Que el ángulo sea el mismo en ambos ejemplos es debido a que el método **"`GetExteriorAngleDegree();`"** calcula siempre el ángulo exterior que forma la línea de dirección con la **LRY** (indicado en las *Figuras 58* y *59*), por lo que nunca serán negativos. Es necesario por tanto establecer una condición para indicar cuándo se trabaja con ángulos positivos y cuándo con negativos.

Como puede observarse en la *Figura 55,* aquellos ángulos comprendidos entre 0 y 180º en sentido horario son positivos, mientras que los comprendidos entre 0 y 180º en sentido antihorario son negativos. Esto se traduce en el algoritmo de la siguiente manera:

Si la componente "X" del punto **VD** es mayor que la componente "X" del **PM** los ángulos serán positivos, por el contrario si la componente "X" del punto **VD** es menor que la componente "X" del **PM** los ángulos serán negativos. Si son iguales la flecha estará alineada con la línea azul *ver Figura 60.*

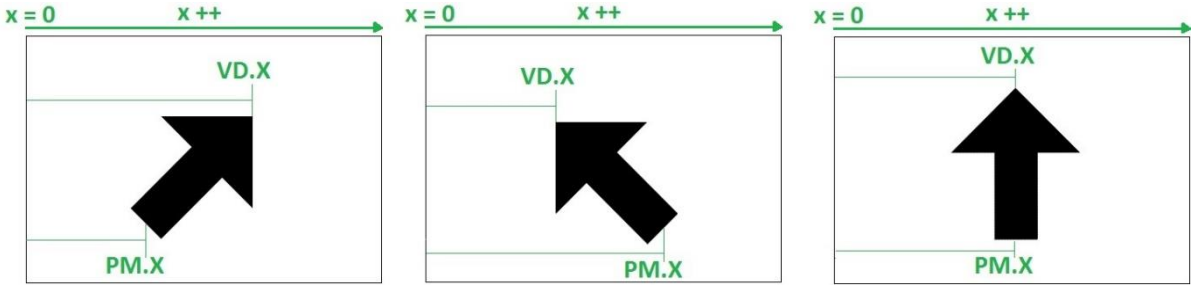


Figura 60. Método para definir el signo de Yaw

Ahora ya sí se han conseguido calcular los ángulos de Yaw con sus signos correspondientes, estos ángulos con su signo se guardarán en una variable para poder trabajar con ellos en siguientes fases del algoritmo.

En los siguientes apartados se van a explicar diferentes arreglos que se han programado en el algoritmo con el fin de hacer el código lo más robusto posible.

4.4.4.3 Métodos para hacer el guiado más robusto

El dron es capaz de ejecutar un giro continuo porque a cada frame el dato de Yaw es coherente con el anterior. Sin embargo, hay varios factores que pueden provocar que entre un frame y el siguiente los datos de Yaw sean muy dispares provocando que el dron gire de forma errónea y descontrolada. A continuación se van a exponer los factores que provocan esto (fallo en el reconocimiento de la flecha y detección de varias flechas en un mismo frame) y también la forma de solucionarlo que se ha implementado en el algoritmo.

- **Fallo en el reconocimiento de la flecha:**

Se va a suponer el siguiente caso: El algoritmo ha detectado y reconocido correctamente una flecha, por lo que calcula su ángulo Yaw correspondiente y el dron comienza a girar. Justo en el siguiente frame captado por la cámara debido a alguna causa puntual como puede ser un cambio en la iluminación ambiente, un ángulo de cámara complicado o cualquier otra causa, el algoritmo sí detecta que está captando un objeto pero no lo reconoce como una flecha. En este frame por tanto no se calculará ángulo de Yaw, estableciéndolo por defecto igual a cero, lo que hará que el dron detenga su giro. En el siguiente frame se han recuperado las condiciones normales, y el algoritmo vuelve a ser capaz de detectar, reconocer y calcular el ángulo de Yaw de la flecha, lo que provoca que el giro vuelva a arrancar.

En resumen debido a algún fallo puntual en el reconocimiento a lo que se denominará “parpadeo” se interrumpe el giro del dron provocando un descontrol y una alteración en el correcto funcionamiento del algoritmo. Si además estos parpadeos se producen varias veces durante el giro las consecuencias en la interrupción de la continuidad son peores. Es por ello que se ha implementado en el algoritmo una forma de solucionar estas interrupciones:

La idea consiste en que si se produce un fallo en el reconocimiento de la flecha se tenga guardado el último ángulo Yaw de esa flecha que se calculó. Así cuando la flecha no sea reconocida se utilizará el ángulo Yaw guardado, que al ser de frames consecutivos será muy próximo al real, lo que permite mantener la uniformidad de giro. Cuando el algoritmo recupere la capacidad de reconocer la flecha el ángulo Yaw

volverá a ser el que se calcule con la detección actual. Cabe comentar que con este procedimiento no existe el riesgo de que se guarde el ángulo de una detección que no sea flecha. El motivo es que desde un primer momento no se habrá detectado ángulo para esa detección por lo que no habrá nada que guardar. Por lo tanto la idea expuesta sólo aplica para detecciones que han superado los filtros del reconocimiento, es decir son flecha, sólo que en un frame esa flecha se ha dejado de reconocer.

Para explicar cómo se traduce esto en el algoritmo hace falta hacer previamente el siguiente inciso:

Para una comprensión más fácil de los procedimientos que se han explicado a lo largo de este capítulo, siempre se ha tratado el caso de que sólo se detecte una única flecha en el frame de la cámara. Sin embargo esto no tiene por qué ser así, también puede ser que se detecten varias flechas en un mismo frame. Para explicar el añadido en el algoritmo que supone esto, se van a recuperar dos aspectos que van a tener gran influencia en la solución que se propone en este apartado: El primero de ellos es que se tiene una variable booleana que toma valor verdadero si la detección ha superado correctamente todos los filtros en la fase de reconocimiento, *ver final del apartado 4.4.3.2.* En realidad no se tiene sólo una variable booleana, sino también un "vector de booleanas" **VB** donde se almacena el valor que toma la variable booleana para cada flecha presente en el frame. El segundo aspecto es que se tiene una variable donde se almacena el ángulo de Yaw calculado, *ver final del apartado 4.4.4.2.* De la misma manera no se tiene sólo una variable, sino que se dispone además de un vector **VA** donde se van a guardar los ángulos Yaw de cada flecha presente en el frame.

En cada frame captado por la cámara se analizan todas las detecciones presentes en ese frame. Aquellas que no superen los dos primeros filtros (número de vértices y área mínima) no tendrán asignada variable booleana ni tampoco se calculará su ángulo Yaw. A aquellas que sí hayan superado los dos primeros filtros se les asigna un valor de la booleana verdadero en caso de que la detección haya superado el tercero y cuarto, y falsa en el caso de que no haya superado alguno de ellos. Cuando la booleana es verdadera se calcula el ángulo Yaw para esa detección. De esta manera si por ejemplo el frame contiene dos flechas, el **VB** y el **VA** contendrán dos datos respectivamente (los valores de las booleanas y los ángulos asociados a cada una de las flechas). Si el frame contiene tres flechas, los vectores contendrán tres datos etc.

Después de este inciso ya se puede explicar el método implementado en el algoritmo para conservar el último ángulo Yaw calculado en caso de fallo momentáneo del reconocimiento de flecha. El método se basa en jugar con el tamaño y posiciones del **VB** y **VA**. Para una mejor comprensión de la solución se van a utilizar varios ejemplos ilustrados con figuras. En todas ellas los recuadros coloreados representan el vector de booleanas, y los que están sin colorear el vector de ángulos:

- *Ejemplo 1 Dos flechas en el frame:*

Este ejemplo contempla el caso de que dentro de un mismo frame se detecten dos flechas. En la columna del primer frame del ejemplo de la *Figura 61* comienza analizándose la primera flecha. Ésta ha superado todos los filtros de reconocimiento por lo que el valor de la booleana asignada a ella será verdadera, y se calculará su ángulo Yaw correspondiente. Después se comienza a analizar la segunda flecha presente en el frame, a la que también se le asigna una variable verdadera por haber

superado los filtros del reconocimiento y se procede a calcular su ángulo. Cabe observar que cuando se comienza a analizar la segunda flecha, el **VA** ya tiene guardado el ángulo Yaw correspondiente a la primera flecha (60º). Finalmente como ya no hay más flechas presentes en el frame el resultado final de los vectores sería el mostrado bajo la línea horizontal. El **VB** tendrá almacenado dos variables verdaderas, y el **VA** los valores de Yaw calculados para cada flecha (60º y 30º).

En el segundo frame se observa que en la primera flecha se ha producido uno de los fallos de reconocimiento de los que se hablaba al inicio de este apartado, por lo que la variable booleana asignada a ella será falsa. Si se observa el **VA** se ve que tiene guardados los valores que se obtuvieron en el frame anterior. Como para el caso de la primera flecha no se va a calcular un nuevo ángulo, se va a conservar el valor del frame anterior (60º) y se va a proceder a analizar la segunda flecha contenida en el frame. La segunda sí consigue superar los filtros del reconocimiento por lo que se procede a reemplazar el valor del ángulo del frame anterior por el actual. Como ya no hay más flechas en el frame el resultado final de ambos vectores se muestra bajo la línea horizontal.

En el tercer frame el algoritmo otra vez vuelve a reconocer la primera flecha por lo que se le asigna el valor booleano verdadero y se reemplaza el ángulo que se conservó del primer frame por el ángulo actual. En cuanto a la segunda flecha también supera el reconocimiento y reemplaza el valor de ángulo del frame anterior por el nuevo. El resultado final de ambos vectores se encuentra bajo la línea horizontal.

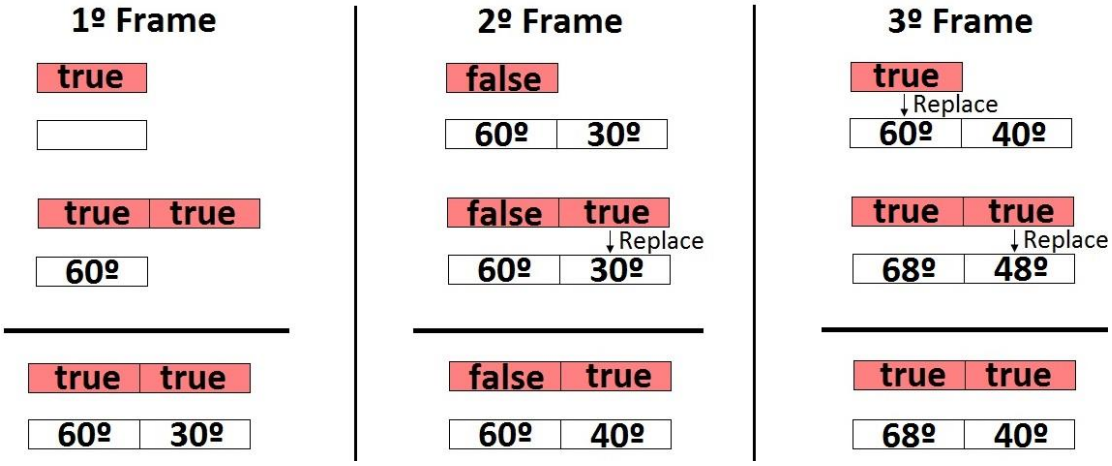


Figura 61. Ejemplo 1: Dos flechas en el frame

- *Ejemplo 2 Dos flechas en los primeros frames, después se añade una tercera*

En este ejemplo se ilustra que no siempre los frames contienen el mismo número de flechas. En este caso concreto se empieza captando un número de flechas determinado, para después detectar alguna más. Esto puede ocurrir cuando el dron esté avanzando para detectar la siguiente flecha y por el angular de la cámara se detecte esa flecha y la de después.

Para ilustrarlo se va a reutilizar el ejemplo anterior modificando la parte nueva *ver Figura 62*:

La explicación para los dos primeros frames es exactamente la misma que en el ejemplo anterior, el tercer frame es diferente porque en este caso va a aparecer una tercera flecha: En el tercer frame se empieza analizando la primera de las flechas que supera el reconocimiento, se establece para ella un valor de booleana verdadera y se reemplaza el ángulo del frame anterior con el nuevo ángulo calculado, procediendo de la misma manera para la segunda flecha. A continuación es el turno de analizar la flecha que ha aparecido nueva, ésta flecha también supera el reconocimiento por lo que se procede a calcular su ángulo Yaw. En este caso no se reemplaza ningún valor del **VA** porque como en el frame anterior sólo se detectaron dos flechas su capacidad era de dos, lo que se hace en su lugar es ampliar el **VA** una posición más para guardar el ángulo de la nueva flecha que ha aparecido. El resultado final de ambos vectores se encuentra bajo la línea horizontal.

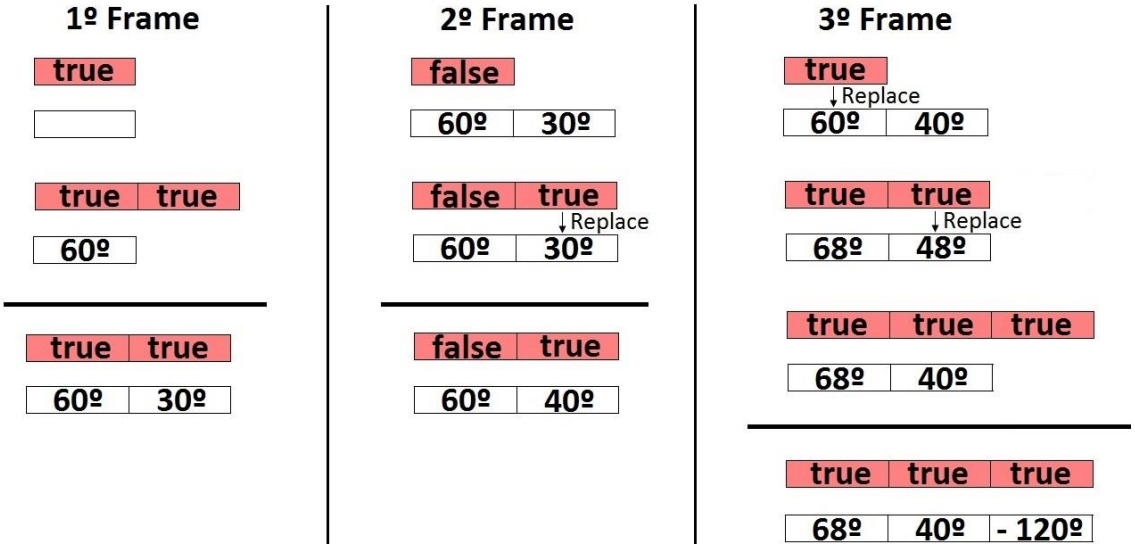


Figura 62. Ejemplo 2: Dos flechas en los primeros frames, después se añade una tercera

- *Ejemplo 3: Dos flechas en los primeros frames, después sólo una*

El Ejemplo 3 ilustra el caso contrario al segundo ejemplo, al principio se están detectando un número determinado de flechas para después pasar a detectar menos. Esto puede ocurrir cuando el dron esté avanzando para detectar la siguiente flecha y por el angular de la cámara se deje atrás la detección de la flecha anterior que ya se analizó.

Para ilustrarlo se va a reutilizar el ejemplo anterior modificando la parte nueva *Ver Figura 63*:

La explicación para los dos primeros frames es exactamente la misma que en el ejemplo anterior, el tercer frame es diferente porque en este caso va a desaparecer la detección de una de las flechas: En el tercer frame sólo se está detectando una flecha, que supera los filtros de la fase de reconocimiento asignándole un valor de booleana verdadero. Después se calcula su ángulo Yaw que reemplazará el ángulo correspondiente al frame anterior. Una vez hecho esto no hay más detecciones de flecha, los vectores resultantes se muestran bajo la línea horizontal.

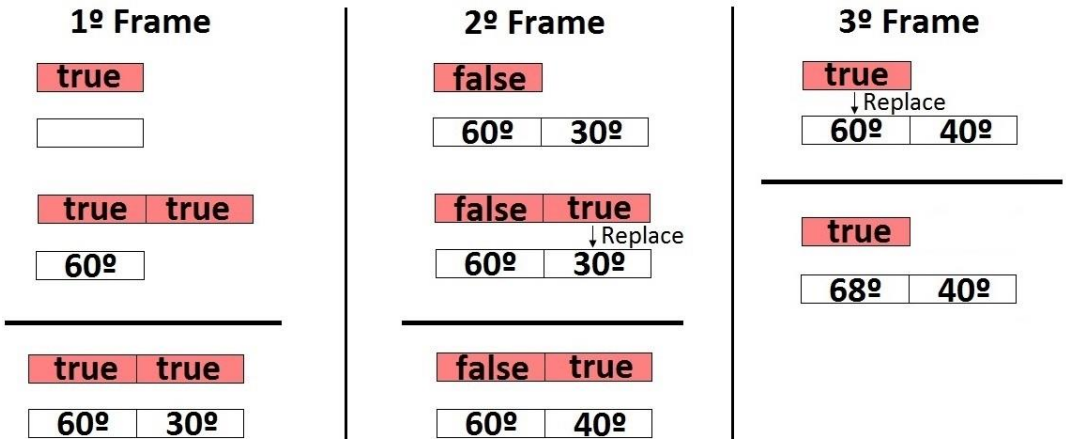


Figura 63. Dos flechas en los primeros frames, después sólo una

Como puede observarse en la *Figura 63* en el tercer frame el **VB** tiene capacidad uno, y está almacenando el valor de booleana asignado a la flecha analizada. En cuanto al **VA**, la primera posición almacena el valor del ángulo calculado para la flecha presente en el tercer frame, sin embargo hay un valor más guardado, el de la segunda flecha que se calculó en el frame anterior. Como no ha habido un número de flechas superior al del anterior frame ese valor no se ha podido reemplazar y se ha quedado guardado. Para solucionar esto se va a establecer la siguiente condición en el algoritmo: Si cuando se han terminado de analizar todas las detecciones presentes en el frame la capacidad del **VB** es menor que la del **VA** se irá borrando la última posición de éste último hasta que ambas capacidades sean iguales. Para obtener las capacidades de los vectores se utilizará la propiedad **`".Count",`** para borrar una única posición del vector se utiliza el método **`".Remove()"`** y para acceder a la última posición del vector se utiliza el método **`".Last()"`**

Cabe comentar que se tiene otro vector llamado "Vector de Vértices de Dirección" **VVD** donde se almacenan las posiciones de los **VD** de todas las flechas presentes en el frame. Para este vector se sigue exactamente el mismo procedimiento que el explicado con el **VA**, si falla el reconocimiento en algún frame se mantiene guardada la posición del **VD** del frame anterior.

La capacidad del **VVD** siempre será igual a la del **VA** ya que ambas capacidades vienen dadas por el número de flechas presentes en el frame. Además al rellenarse de forma simultánea sus posiciones coincidirán, es decir, el **VD** almacenado en la posición uno del **VVD** es el correspondiente a la flecha cuyo ángulo Yaw está guardado en la posición uno del **VA** y así sucesivamente.

El **VVD** va a ser de utilidad en el siguiente caso.

- **Detección de varias flechas en un mismo frame:**

Cuando aparecen varias flechas en un mismo frame debe imponerse una norma para decidir cuál de los ángulos que indican se utilizará para guiar al dron.

Se pretende utilizar el ángulo Yaw correspondiente a las nuevas detecciones, e ignorar aquellos que ya se han transmitido con anterioridad hasta que desaparezcan del plano. Como el dron avanza hacia delante, las nuevas detecciones aparecerán siempre por la parte superior de la imagen que capta la cámara, mientras que las

detecciones antiguas estarán en posiciones inferiores hasta que desaparezcan del plano. Traducido en coordenadas, las flechas nuevas se encontraran en coordenadas "Y" de valores inferiores a las antiguas *ver Figura 64:*

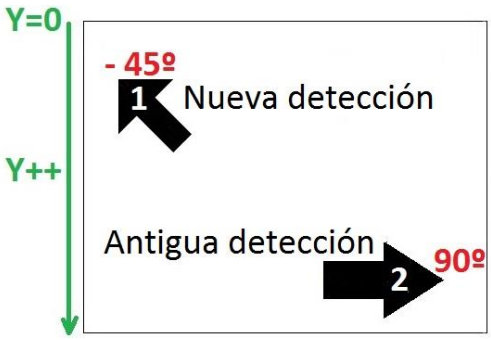


Figura 64. Ejemplo de detección nueva y antigua en el caso de que varias flechas se encuentren presentes en el mismo frame

Por lo tanto el procedimiento que se va a seguir en el algoritmo para detectar cuál de las flechas presentes en el frame corresponde a la detección más reciente será ir comparando las componentes "Y" de su **VD**. Aquella flecha cuyo **VD** tenga una componente "Y" más pequeña se corresponderá con la detección más reciente, por tanto será su ángulo Yaw el que debe utilizarse para guiar al dron *ver Figura 65.*

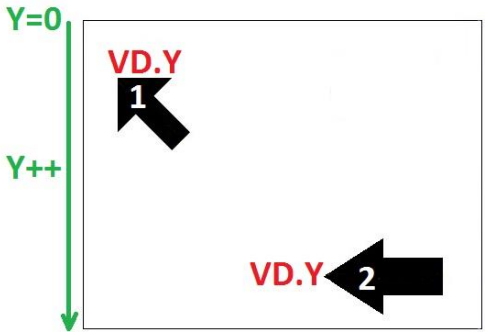


Figura 65. Ejemplo de comparación de las componentes "Y" de los vértices de dirección de dos flechas

Para efectuar la comparación hay que hacer uso del **VVD**, que contiene los **VD** de todas las flechas presentes en el frame. Además también se necesitan dos variables auxiliares, una denominada "Flecha Reciente" **FR** para guardar el valor de la componente "Y" del **VD** que haya resultado menor hasta el momento, y otra llamada "Índice de Vector" **Iv** para guardar el valor de la posición que ocupa ese **VD** en el **VVD**.

De tal manera que si la componente "Y" del **VD** actual es menor que la almacenada en la variable auxiliar **FR**, se sobrescribirá la variable auxiliar con el valor de la actual y además en la variable **Iv** se guardará la posición que ocupa en el **VVD** el vértice actual. Se procederá de la misma manera hasta que se hayan comparado todas las flechas, momento en el cual se tendrán guardadas en las variables auxiliares los valores de la componente "Y" del **VD** correspondiente a la detección de flecha más reciente, y la posición en la que está almacenado ése vértice en el **VVD**.

Debido a la propiedad del vector de vértices, el ángulo Yaw con el que se guiará al dron será aquel que en el **VA** se encuentre en la posición marcada por la variable auxiliar **Iv**.

Después de esta explicación ya se ha conseguido localizar la detección de flecha más reciente, y por tanto será su ángulo Yaw el que se transmita al dron. Pero antes de dar este apartado por concluido se va a plantear una problemática:

El método explicado también es susceptible a los "parpadeos". Para comprenderlo mejor se va a ilustrar con el ejemplo de la *Figura 66:*

En la parte izquierda de la *Figura 66* se observa que el frame contiene dos flechas, ambas se están detectando y reconociendo correctamente por lo que cada una tienen un valor Yaw asociado. Gracias al método que se acaba de explicar, el ángulo Yaw que se va a transmitir al dron será el de la última flecha detectada, es decir el de la flecha número 1 cuyo Yaw es de 45º. En el siguiente frame, ilustrado en la mitad de la *Figura 66,* debido a una mala segmentación, un cambio de iluminación ambiente o cualquier otra causa, se ha perdido la detección de la flecha número 1, por lo que el valor de Yaw que se transmitirá al dron será el de la flecha más alta del frame en ese momento, que como la número 1 no se está detectando pasa a ser la flecha número 2, cuyo Yaw es de 90º (valor que no debería transmitirse puesto que es una flecha antigua que ya se analizó). Como se observa en la parte derecha de la *Figura 66*, en el frame posterior se vuelve a recuperar la detección de la flecha número 1 que se había perdido, por lo que se volverá a enviar al dron su ángulo Yaw, 45º.

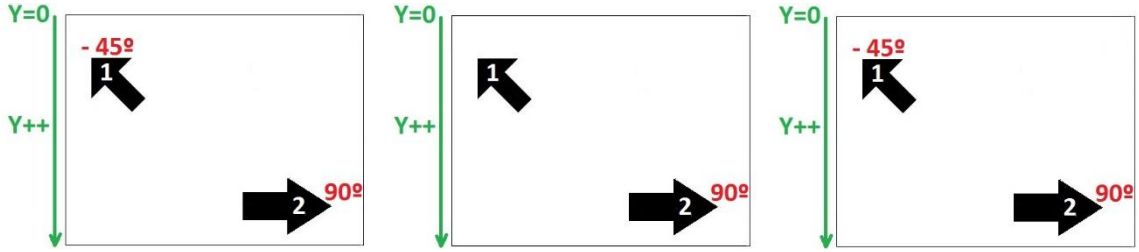


Figura 66. Ejemplo de "parpadeo" por fallo en la detección de la flecha más alta del frame

El resultado de este ejemplo es que se ha transmitido al del dron la siguiente secuencia de ángulos "- 45º, 90º, - 45º". Para empezar, que se haya transmitido el ángulo 90º es un error porque es el asociado a una flecha antigua que ya se analizó. Además al ser ángulos muy dispares, provocan cambios de dirección en el giro muy bruscos. Para solventar este aspecto de los "parpadeos" por fallo en la detección de la flecha más alta del frame se ha programado en el algoritmo lo siguiente:

Lo que se va a hacer es comparar la componente "Y" del **VD** correspondiente a la última flecha detectada en el frame actual, con la componente "Y" del **VD** correspondiente a la última flecha detectada del frame anterior. Como el dron avanza hacia delante, la componente "Y" del **VD** del frame actual será ligeramente mayor que la del frame anterior, pero no mucho más. Se establece un intervalo de 100 píxeles de margen. Esto se va a explicar con el ejemplo de la *Figura 67:*

En el primer frame se tienen dos flechas contenidas en el plano de la imagen que capta la cámara, se procede a guardar el valor de la componente "Y" del **VD** correspondiente a la última flecha detectada que en el caso del ejemplo es la número uno, ya que su **VD** tiene una componente "Y" menor que la del **VD** de la flecha número dos.

En el segundo frame se ha perdido la detección de la flecha número uno por lo que sólo está siendo detectada la flecha número dos. El algoritmo busca el **VD** cuya componente "Y" sea menor dentro de todas las detecciones de flechas contenidas en el plano, que se corresponderá con el de la flecha número dos, es decir, toma ésta flecha como si hubiera sido la detección más reciente. Ahora se comprueba si la componente "Y" del **VD** de la flecha correspondiente a la detección más reciente del frame actual es menos de 100 píxeles mayor que la componente "Y" del **VD** de la última flecha detectada del frame anterior (flecha número uno). En la *Figura 67* se observa gráficamente que esta condición no se cumple, lo que significa que se ha producido en este frame un error de detección. Por ello lo que se hace es mandar al dron el ángulo Yaw correspondiente a la última flecha detectada del frame anterior, 45º, en vez de mandar los 90º del frame dos que se sabe que son erróneos. Como la comprobación no se ha cumplido sigue teniéndose guardado el valor de la componente "Y" del vértice de dirección del frame 1, este dato sólo se actualizará en el caso de que la comprobación se cumpla.

En el tercer frame se ha vuelto a recuperar la detección de la flecha número uno. Se comparan las componentes "Y" de los **VD** de ambas flechas, y se observa que la detección más reciente corresponde a la flecha número uno, ya que su componente "Y" es la menor. A continuación se comprueba si la componente "Y" del **VD** de la última flecha detectada del frame actual es menos de 100 píxeles mayor que la componente "Y" del **VD** de la última flecha detectada del frame anterior (que sigue siendo el frame uno). En la *Figura 67* se observa que esta comprobación se cumple, por lo que el ángulo transmitido al dron será el de la flecha número uno del frame tres. Debido a que la comprobación se ha cumplido, se guarda el valor de la componente "Y" del **VD** de la flecha número uno del frame tres sobrescribiendo el anterior que se tenía guardado.

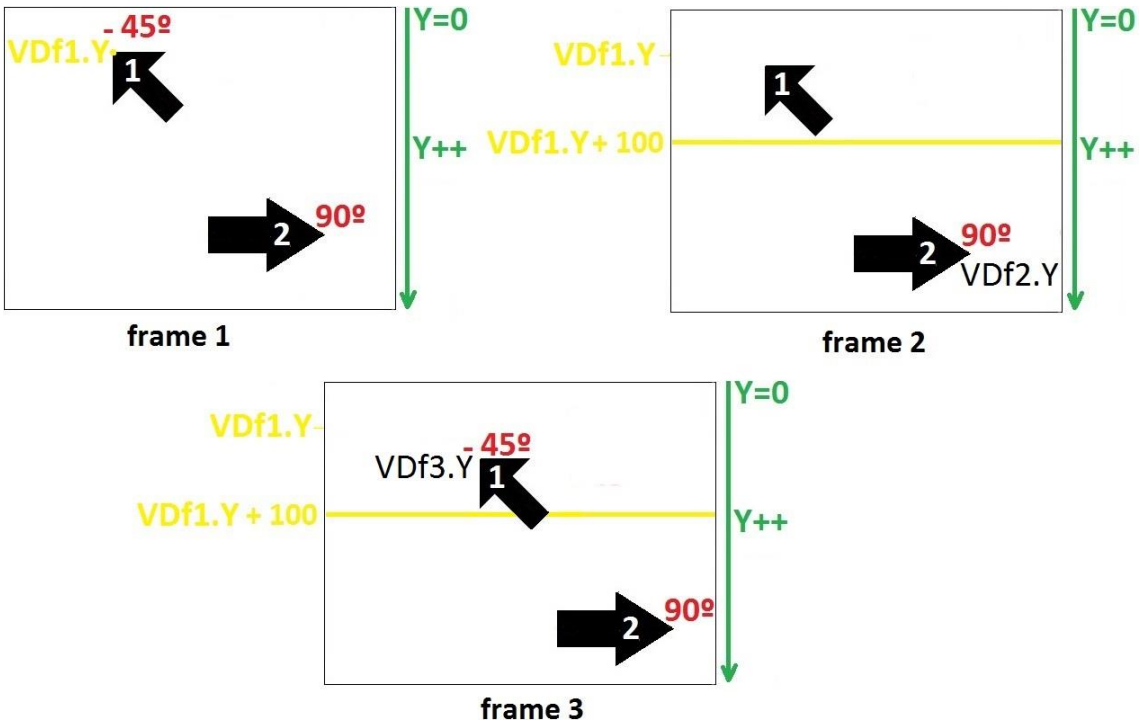


Figura 67. Solución propuesta al ejemplo del "parpadeo" que se produce por un fallo en la detección de la flecha número uno del frame 2

Para hacer el código más robusto se va a realizar el mismo procedimiento para el eje “X” con un intervalo para hacer la comprobación de +/- 100 píxeles.

En conclusión lo que se pretende hacer con el método explicado en este apartado es hacer consciente al algoritmo que entre un frame y el siguiente la detección de flecha tiene que ser coherente, es decir, si en un frame la flecha estaba en una posición determinada al siguiente frame esa misma flecha no puede aparecer en una posición excesivamente distante de la anterior, ya que el dron no se mueve a gran velocidad y el tiempo que transcurre entre un frame y el siguiente es muy pequeño.

4.4.4.4 Control

El control del dron es lo que va a permitir el guiado del mismo durante el recorrido de flechas. El control se va a basar en dos modos de vuelo: “forward” (movimiento hacia delante) y “detection” (giro en función del ángulo que marque la flecha). El dron entra en modo “detection” cuando detecta una flecha y sale de ese modo cuando haya terminado de efectuar el giro correspondiente. Cuando el modo “detection” no está activado, se activa el modo “forward” con la finalidad de moverse hacia delante para buscar la siguiente flecha.

Para verlo más claro se ha incluido la *Figura 68*. En ella, los recorridos representados con una línea verde indican que el dron está navegando en modo “forward”, mientras que los representados con una línea naranja indican que se mueve en modo “detection”

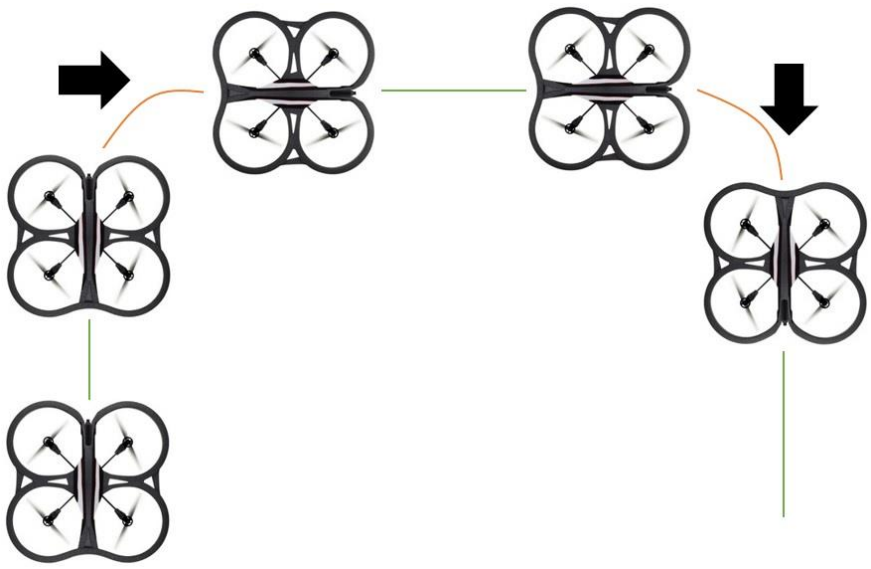


Figura 68. Ejemplo del recorrido de un dron, las líneas verdes representan que viaja en modo "forward" y las naranjas en modo "detection"

Por tanto el dron vuela hacia delante hasta que detecta flecha, momento en el cual girará hasta haber alineado su dirección y sentido con la dirección y sentido de la flecha detectada. Una vez efectuado el giro completo el dron volverá al modo vuelo hacia delante en búsqueda de la siguiente flecha.

Para saber si en el frame actual se ha detectado flecha se hace uso de una variable booleana llamada **“arrow”**. Que **“arrow”** sea verdadera indica que en el frame actual

se ha detectado flecha, mientras que si es falsa es indicativo de que no se ha detectado ninguna flecha en ese frame.

En este apartado se va a comenzar explicando el modo "detection", para después hablar del modo "forward". Finalmente se terminará explicando la manera de transmitir toda la información obtenida durante el algoritmo al controlador del dron para efectuar los movimientos correspondientes.

- **Modo "detection":**

El modo "detection" comienza cuando se ha detectado flecha, que viene indicado por un valor de **"arrow"** verdadero. Cuando esa variable adquiere valor verdadero, comienza el giro del dron.

Durante el giro pueden darse dos casos:

- Caso 1: El dron es capaz de efectuar el giro completo sin perder la flecha del plano de la cámara. El giro comienza y termina con **"arrow"** verdadero.
- Caso 2: Durante el giro la flecha se sale del plano sin que el dron haya terminado de efectuar el giro completo. El giro comienza con **"arrow"** verdadero y termina con **"arrow"** falso. Este caso se divide en dos sub-casos:
    - Caso 2.1: Durante el giro los ángulos que marca el sensor de la brújula del dron siempre son del mismo signo, o positivos o negativos
    - Caso 2.2: Durante el giro se produce un cambio de signo en los ángulos que marca el sensor de la brújula del dron.

Que la flecha se salga del plano de la cámara antes de que el dron haya efectuado el giro completo en el caso 2 supone que ya no se disponga del ángulo que indica la flecha. Esto supone que no se tenga la información de en qué momento debe cesar el giro porque el dron ha conseguido alinearse con la dirección y sentido de la flecha.

En este apartado se van a explicar todos los casos nombrados, y el método que se sigue para saber cuándo parar el giro en el caso 2.

- *Caso 1 Flecha dentro del plano de la cámara:*

En el caso 1 el valor de **"arrow"** va a ser verdadero durante todo el giro. Cuando **"arrow"** es verdadero el ángulo que se va a utilizar para el guiado del dron en el giro es directamente el que indica la flecha en cada momento (Yaw). El ángulo que indica la flecha disminuye conforme el dron alinea su dirección con la que marca la flecha. Cuando las direcciones finalmente quedan totalmente alineadas el ángulo que indica la flecha será cero, momento en el cuál el dron dejaría de girar.

- *Caso 2 Flecha sale del plano de la cámara durante el giro:*

En el segundo caso, el dron comienza detectando flecha siendo el procedimiento a seguir el mismo que en el primer caso. Sin embargo cuando la flecha sale del plano, hay que utilizar otro método para poder terminar el giro, ya que ahora no se dispone de la referencia del ángulo que va marcando la flecha en cada momento para poder enviar esta información al dron. La información para poder tener referencia sobre cuánto lleva girado el dron la va aporta el sensor de la brújula que lleva incorporado, al que se accede mediante el comando **"data.psi"**. La referencia del sensor brújula se indica en la *Figura 69*:

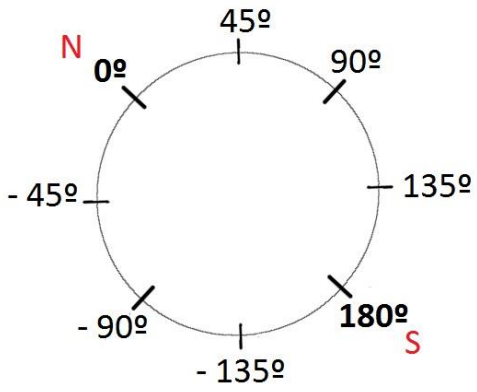


Figura 69. Referencia del sensor brújula

- Caso 2.1 No se produce cambio de signo en los valores que marca la brújula:

A continuación se contempla el caso de que durante todo el giro no se produzca un cambio de signo en los ángulos que marca la brújula. Por tanto el proceso de cálculo de ángulo de Yaw es el siguiente: Se van a guardar en dos variables el último ángulo que marcó la flecha antes de salirse del plano, y el valor que indicaba la brújula en ese momento. El ángulo se guarda en una variable denominada **"reference"** ya que su contenido se va a usar como referencia (indica cuánto le queda al dron por girar para efectuar el giro completo). Mientras que el valor que indicaba la brújula se va a guardar en una variable llamada **"PsiAddition"**. Una vez guardados estos valores se va a sumar lo contenido en ambas variables, guardando el resultado en una tercera llamada **"Addition"** *ver Ecuación 4.26*:

$$Addition = reference + PsiAddition \qquad \textbf{(4.26)}$$

El dron deberá dejar de girar cuando la brújula marque el valor indicado por **"Addition"**. En cada frame el dron debe recibir información sobre un valor de Yaw, equivalente al que recibiría si la flecha siguiera dentro del plano de la cámara. Para calcularlo se le resta a la variable **"Addition"** el valor del ángulo que marca la brújula en cada frame *ver Ecuación 4.27:*

$$Yaw = Addition - data.Psi \qquad \textbf{(4.27)}$$

Para una mejor comprensión de lo que se ha explicado se va a incluir el siguiente ejemplo numérico:

En el momento en el que la flecha se sale del plano se guardan los siguientes valores: "$reference = 30\,º$" (lo que indica que al dron le quedan por girar 30º para alinear su dirección y sentido con el de la flecha) y "$PsiAdddition = 20º$". Al sumar ambas variables se obtiene *ver Ecuación 4.28*:

$$Addition = 30º + 20º = 50º \qquad \textbf{(4.28)}$$

Es decir cuando la brújula marque 50º el dron habrá efectuado el giro completo. Como se verá al final de este apartado, al ser el valor de referencia positivo el dron va a girar en sentido horario. Por tanto los ángulos Yaw en cada frame son los siguientes, (por simplificación en este ejemplo se ha supuesto que a cada frame el dron avanza diez grados) *ver Ecuaciones 4.29, 4.30, 4.31 y 4.32*:

Frame 1: $Yaw = 50 - 20 = 30$ ***(4.29)***

Frame 2: $Yaw = 50 - 30 = 20$ ***(4.30)***

Frame 3: $Yaw = 50 - 40 = 10$ ***(4.31)***

Frame 4: $Yaw = 50 - 50 = 0$ ***(4.32)***

La información de estos valores de Yaw serán enviados al dron para efectuar el giro. Cuando Yaw es igual a cero significa que el dron ya ha efectuado el giro completo y el dron frenará.

- Caso 2.2 Se produce cambio de signo en los valores que marca la brújula:

La *Ecuación 4.27* es válida siempre que los valores de la brújula se hayan mantenido con el mismo signo, o todos positivos o todos negativos. Si durante el giro del dron se ha producido un cambio de signo en los ángulos que indica la brújula, *ver Figura 69,* la variable **"Addition"** se calcula de forma diferente:

Se guarda en una variable llamada **"YawNS"** el último valor de Yaw calculado antes del cambio de signo, y en una variable llamada **"PreviousPsi"** el último valor que marcó la brújula antes de pasar a marcar los ángulos de signo contrario. **"Addition"** es el resultado de restar ambas variables *ver Ecuación 4.33*:

$$Addition = YawNS - PreviousPsi \quad \textbf{(4.33)}$$

Para entender mejor esta explicación se va a incluir el siguiente ejemplo numérico:

En el momento en el que la flecha se sale del plano se guardan los siguientes valores: "$reference = 30\,º$" (lo que indica que al dron le quedan por girar 30º para alinear su dirección y sentido con el de la flecha) y "$PsiAdddition = 170º$". Al sumar ambas variables se obtiene *ver Ecuación 4.34*:

$$Addition = 30º + 170º = 200º \quad \textbf{(4.34)}$$

Como se verá al final de este apartado, al ser el valor de referencia positivo el dron va a girar en sentido horario. Por tanto los ángulos Yaw en cada frame son los siguientes, (por simplificación en este ejemplo se ha supuesto que a cada frame el dron avanza diez grados) *ver Ecuaciones 4.35 y 4.36*:

Frame 1: $Yaw = 200 - 170 = 30$ ***(4.35)***

Frame 2: $Yaw = 200 - 180 = 20$ ***(4.36)***

Hasta aquí se ha utilizado la fórmula para calcular **"Addition"** de la *Ecuación 4.26*, porque no se ha producido ningún cambio de signo en los valores que marca la brújula. Sin embargo como se puede observar en la *Ecuación 4.36* el último valor que ha marcado la brújula ha sido 180º y el dron aún no ha terminado de girar (le quedan 20º para completar el giro). Por tanto en el siguiente frame se va a producir un cambio de signo y habrá que utilizar la *Ecuación 4.33* para calcular el nuevo **"Addition"** *Ver Ecuación 4.37:*

$$Addition = 20 - 180 = -160 \quad \textbf{(4.37)}$$

Este nuevo valor indica que cuando la brújula marque -160º el dron habrá efectuado el giro completo, a partir de aquí el cálculo de Yaw es el mismo que en el caso 2.1*, ver Ecuaciones 4.38 y 4.39*:

$$\text{Frame 3: } Yaw = -160 - (-170) = 10 \quad \textbf{(4.38)}$$

$$\text{Frame 4: } Yaw = -160 - (-160) = 0 \quad \textbf{(4.39)}$$

- **Modo "forward":**

Como se explicó en el *apartado 4.4.4.4,* el modo "forward" se activa cuando se desactiva el modo "detection", y el "modo detection" se activa cuando se detecta una flecha. Para saber si se ha detectado flecha o no se hace uso de la variable **"arrow"**, que toma valores verdaderos si se está detectando flecha y falsos en caso contrario. Sin embargo, como se ha visto, durante el modo "detection" la variable **"arrow"** no tiene por qué ser verdadera durante todo el giro. Puede que mientras el dron gira la flecha se haya salido del plano de la cámara y **"arrow"** haya pasado a ser falsa estando aún activo el modo "detection" hasta que el dron complete el giro. Es por ello que no se puede utilizar la variable **"arrow"** para efectuar el cambio de modo, ya que con ella no hay manera de distinguir si realmente no se está detectando flecha y debería encenderse "forward" para moverse hacia delante en busca de la siguiente, o no se está detectando flecha porque durante el giro se ha salido del plano de la cámara. En ese caso habría que esperar a que el dron complete el giro antes de pasar a modo "forward".

Se necesita por tanto definir una nueva variable que indique cuándo se pude cambiar de un modo a otro sin ninguna incertidumbre, esa variable se denominará **"turn"**. **"turn"** va a permanecer verdadera desde que el dron ha captado flecha, y por lo tanto ha iniciado el giro, hasta que ha efectuado el giro completo, momento en el cual pasará a ser falsa. Por tanto el modo "forward" estará activo siempre que la variable **"turn"** tenga valor falso.

- **Transmisión de información al controlador:**

Una vez explicado cada modo se va a exponer cómo se transmite la información al dron para que realice los movimientos correspondientes. Para que el dron se mueva, es necesario jugar con los ángulos de navegación, y la forma de conseguirlo es variando la velocidad de las hélices, para ello se utiliza la función **"Navigate(rollValue, pitchValue, yawValue, gazValue);"**. En esta función cada una de las entradas equivale a un ángulo de navegación del dron. Los valores que se pueden introducir son los contenidos en el intervalo entre -1 y 1, equivaliendo el valor 1 a la máxima potencia de los motores que mueven las hélices, el valor 0,5 a la mitad de potencia y así sucesivamente. El signo positivo o negativo indicará cosas diferentes dependiendo de cada ángulo *ver Tabla 3*:

Tabla 3. Signos de los inputs de Navigate y su significado para cada ángulo

	Roll	Pitch	Yaw	Gaz
Positivo	Movimiento hacia la derecha	Movimiento hacia atrás	Giro horario	Ascenso
Negativo	Movimiento hacia la izquierda	Movimiento hacia delante	Giro antihorario	Descenso

Por tanto, en el modo de vuelo "forward" se introducirá un valor en el input de pitch de la función **"Navigate(rollValue, pitchValue, yawValue, gazValue);"** que determinará la velocidad de avance, mientras que todos los demás inputs permanecerán con valor cero para obtener un avance en línea recta.

El valor elegido será constante e igual a -0.2, ya que tras varias pruebas ese valor es que ha permitido un mejor funcionamiento del algoritmo. El signo negativo hace que el dron avance hacia delante.

Cuando se detecta una flecha el valor de pitch pasará a ser cero, por lo que el movimiento hacia delante parará y comenzará el de giro dentro del modo "detection".

En el modo "detection" se enviará un valor en el input de Yaw de la función **"Navigate(rollValue, pitchValue, yawValue, gazValue);"** que determinará la velocidad de giro, mientras que todos los demás inputs permanecerán con valor cero para que el único movimiento del dron en ese momento sea de giro.

El valor elegido es constante e igual a 0.5 (en valor absoluto) lo que significa que el giro se efectúa a la mitad de potencia de los motores. Se ha elegido ese valor ya que tras varias pruebas es el que ha permitido un mejor funcionamiento del algoritmo. El signo se va a asignar en función del ángulo que marca la flecha: si el ángulo es positivo, el signo será positivo lo que hace que el giro se efectúe en sentido horario, mientras que si el ángulo de la flecha es negativo el signo será negativo efectuando el giro en sentido antihorario.

Debido al valor elegido 0.5, el giro se efectúa a la mitad de la potencia de los motores. Lo que ocurre es que debido a la velocidad de giro que marca el valor elegido, es muy difícil que el dron pueda frenar totalmente si espera a que la flecha indique ángulo 0º. Un ejemplo de esto se encuentra en la *Figura 70:*

En el frame anterior la flecha marcaba 3 º y debido a la velocidad de giro, en el frame actual la flecha marca - 3º. Es decir de un frame a otro el dron ha recorrido seis grados, y no ha captado el momento en el que la flecha indicaba 0º.

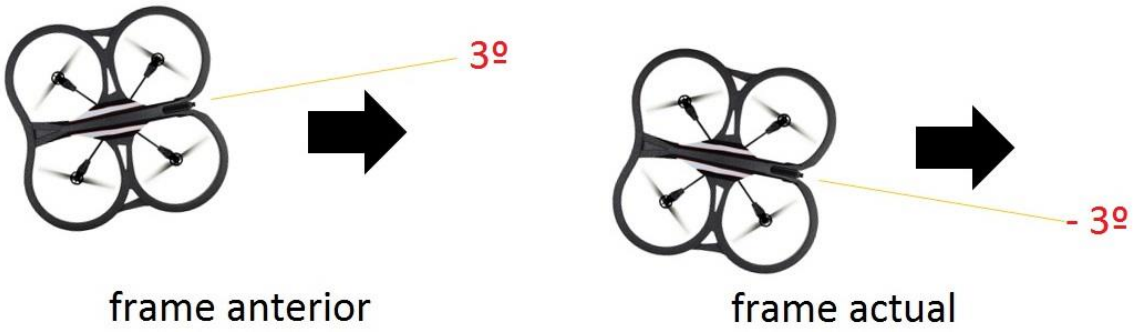


Figura 70. Ejemplo de los ángulos que le llegan al controlador en el giro

La solución para poder frenar el giro adecuadamente es establecer un intervalo que acorde al valor elegido. De tal manera que cuando el dron entre en ese intervalo de ángulos empiece a frenar. En el caso del algoritmo se ha establecido un intervalo de +/- 10º, es decir, cuando el giro del dron llega a valores más bajos que 10 º para el caso de ángulos positivos, o a valores mayores que – 10º para el caso de ángulos negativos, se envía a la función **"Navigate(rollValue, pitchValue, yawValue,**

`gazValue);”` un valor de Yaw igual a cero para frenarlo. Esto significa que el dron conseguirá frenar más fácilmente, pero con un margen de error de +/-10º.

4.5 Alternativas de diseño

Durante la fase de diseño del algoritmo que finalmente se ha propuesto como solución, desarrollado en los *apartados 4.2, 4.3* y *4.4,* han surgido varias alternativas de diseño que finalmente no fueron escogidas para la solución final.

Las principales alternativas se barajaron en el método en el que se iba a basar la detección y reconocimiento de las flechas. Todas ellas están desarrolladas en el *Capítulo 2* junto con las razones de por qué no fueron finalmente escogidas. No obstante a continuación se va a hacer un resumen de esa justificación:

Debido a la problemática que se quiere resolver *ver apartado 1.2,* las principales características que se busca que tenga la solución final son:

1. Detección de objetos de diferentes tamaños (invariabilidad a la escala)
2. Detección de objetos en diferentes orientaciones (invariabilidad a la rotación)
3. Detección de un mismo objeto con variación en su forma (detección de varios tipos de flecha)
4. Bajo tiempo de procesamiento
5. Trabajar a tiempo real

Los principales métodos alternativos a la solución propuesta para la detección de objetos son SURF, SIFT y clasificadores como el tipo Haar. Las conclusiones a las que se llegó al comparar los diferentes métodos en el *Capítulo 2* fueron las siguientes:

Los clasificadores pueden llegar a ser invariantes a la rotación y escala con gran esfuerzo de procesamiento, además para que sean capaces de detectar un mismo objeto con variaciones en su forma (lo que serían diferentes tipos de flecha) habría que entrenar un clasificador para cada una de ellas, lo que supone esfuerzo de procesamiento y limitar en gran medida la cantidad de tipos de flecha que pueden detectar.

Por otro lado SURF y SIFT sí presentan buena invariabilidad a la rotación y escala pero para detectar diferentes tipos de flecha necesitarían servirse de una base de datos con numerosas plantillas del objeto a buscar, en este caso flechas de diferentes tipos. Utilizar bases de datos supone ampliar en gran medida el tiempo empleado en el procesamiento, además de limitar los tipos de flecha que se pueden detectar. Por otra parte estos dos métodos funcionan de forma muy pobre con objetos simples, ya que pueden sacar muy poca cantidad de puntos característicos, por lo que no se obtendría una buena ejecución con el objeto que se quiere detectar en este trabajo, las flechas.

Que estos métodos supongan aumentar el tiempo de procesamiento para poder analizar sus bases de datos es nefasto ya que precisamente lo que se busca es lo contrario, emplear el menor tiempo posible en el procesamiento del algoritmo para poder ganar la carrera de drones autónomos.

En la *Tabla 4* se muestra una comparación de estos métodos con el propuesto en la solución final que se presenta en este trabajo, para cada una de las características básicas que se necesitan alcanzar para la resolución del objetivo.

Tabla 4. Comparación de métodos alternativos con la solución final

Métodos	Invariabilidad a la escala	Invariabilidad a la rotación	Detección de un mismo objeto con variación en su forma	Velocidad de procesamiento	Trabaja a tiempo real
Clasificadores	Sí (con esfuerzo de procesamiento)	Sí (con esfuerzo de procesamiento)	Muy limitados (entrenando un clasificador por cada tipo)	Lenta	Sí
SURF/SIFT	Sí (muy pobre con objetos simples)	Sí (muy pobre con objetos simples)	No con una única plantilla, sí con base de datos	Rápida con una única plantilla, lenta con base de datos	Sí
Solución final	Sí	Sí	Casi ilimitados	Rápida	Sí

Como se puede ver la solución propuesta cumple cada una de las características de forma mucho más óptima que los métodos alternativos. Es invariante a la rotación y escala, es capaz de detectar tipos de flecha casi ilimitados, trabaja a tiempo real y la velocidad de procesamiento es considerablemente más rápida al no basarse en bases de datos (todos estos aspectos van a quedar justificados en el *Capítulo 5*). Por todos estos motivos se eligió el método empleado en la solución final, desarrollado en el *apartado 4.4*, y se descartaron estos métodos alternativos.

Por otro lado también surgió alguna alternativa de diseño en cuanto a qué filtros emplear en la fase de reconocimiento. Al principio del diseño el filtro número 3 era diferente al que se ha empleado finalmente, simplemente se establecía la condición de que si un ángulo de la **ZT** era agudo, y un ángulo de la **ZR** era recto se daba por bueno que la detección era flecha. Esto suponía que superaran el filtro gran cantidad de detecciones que no eran flecha, y que además la dirección de la flecha muchas veces se calculara de forma errónea, ya que de este filtro depende identificar cuál de los dos vértices de la flecha que se encuentran a mayor distancia es el **VD** y cuál el **Vff** *ver Figura 71.* Por esta razón se eligió el filtrado expuesto en el *apartado 4.4.3.2.*

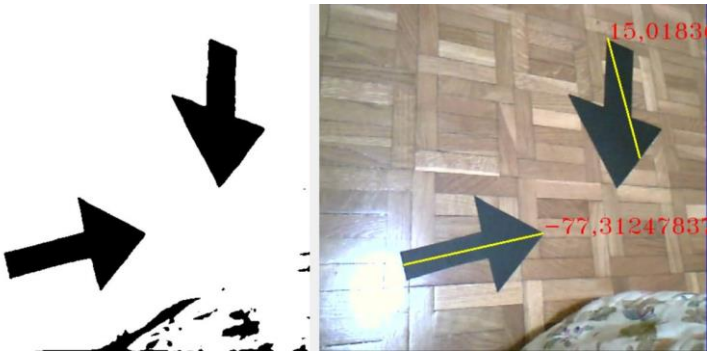


Figura 71. Filtro 3 alternativo: Error en el cálculo de la dirección de la flecha

En conclusión se ha elegido el diseño de la solución final porque es el que ha permitido alcanzar de forma más óptima las cinco características básicas que han de conseguirse para la ejecución del objetivo (invariabilidad a la rotación, invariabilidad a la escala, detección de un mismo objeto con variación en su forma, velocidad de procesamiento y trabajo en tiempo real), y porque es el que ha permitido obtener mejores resultados, ampliados en el *Capítulo 5.*

Capítulo 5: Trabajo experimental y resultados

5.1 Introducción

En este apartado se van a detallar los resultados obtenidos en la fase de pruebas del algoritmo que se propone como solución final, y hasta qué punto se ha conseguido alcanzar el objetivo de este trabajo. Para ello primero se va a exponer y analizar los resultados obtenidos en cada una de las características implementadas para la resolución de la problemática planteada en este trabajo, y después los resultados correspondientes a la fase de guiado.

Cabe comentar que en las figuras que se van a mostrar en cada apartado se produce detección y reconocimiento de flecha de forma correcta cuando se encuentran delineadas por líneas de color morado y tienen calculada su dirección, que viene representada con una línea amarilla.

5.2 Características

En este apartado se van a mostrar y analizar los resultados que demuestran hasta qué punto se han conseguido alcanzar las características necesarias para la resolución de la problemática expuesta *en el apartado 1.2* y la consecución de los objetivos recogidos en el *apartado 1.3.*

Este apartado se va a dividir en dos, el de las características básicas necesarias para que el dron sea capaz de recorrer el camino de flechas con éxito, y otro denominado características extra donde se va a hablar los colores de flechas que se pueden detectar.

5.2.1 Características básicas

Las características básicas que se necesitaban implementar para la consecución del objetivo son:

1. Detección de objetos de diferentes tamaños (invariabilidad a la escala)
2. Detección de objetos en diferentes orientaciones (invariabilidad a la rotación)
3. Detección de un mismo objeto con variación en su forma (detección de varios tipos de flecha)
4. Bajo tiempo de procesamiento
5. Trabajar a tiempo real

5.2.1.1 Invariabilidad a la escala

El algoritmo propuesto como solución es capaz de detectar flechas de cualquier tamaño, por lo que se ha conseguido invariabilidad a la escala, *ver Figura 72.*

Figura 72. Invariabilidad a la escala

Como se ha visto en el *Capítulo 2* algunos métodos ya existentes como "Template Matching" no son invariantes a la escala a no ser que utilicen bases de datos con plantillas de diferentes tamaños, u otros como "SURF", "SIFT", clasificadores etc sí lo son pero funcionan mal si los objetos a detectar son simples como en el caso de las flechas, y con un esfuerzo de procesamiento mayor que el algoritmo que se propone como solución en este trabajo *ver apartado 5.2.1.4*.

5.2.1.2 Invariabilidad a la rotación

El algoritmo propuesto como solución es capaz de detectar flechas en cualquier orientación, por lo que se ha conseguido invariabilidad a la rotación, *ver Figura 73.*

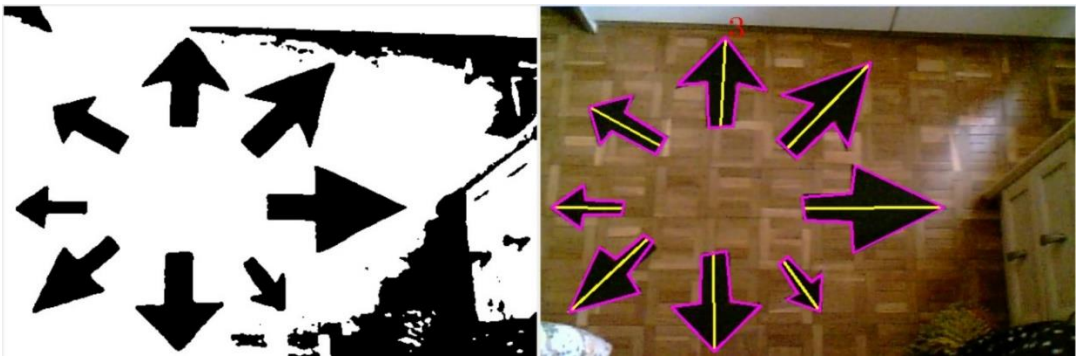


Figura 73. Invariabilidad a la rotación

Como se ha visto en el *Capítulo 2* algunos métodos ya existentes como "Template Matching" no son invariantes a la rotación a no ser que utilicen bases de datos con plantillas en diferentes orientaciones, u otros como "SURF", "SIFT", clasificadores etc. sí lo son pero funcionan mal si los objetos a detectar son simples como en el caso de las flechas, y con un esfuerzo de procesamiento mayor que el algoritmo que se propone como solución en este trabajo como se verá en el *apartado 5.2.1.4*.

5.2.1.3 Detección de un mismo objeto con variación en su forma

Como se indicó en el *apartado 1.2* las bases de la competición establecen que el recorrido puede estar formado por flechas de diferentes tamaños y formas.

La variación en las formas de las flechas va a venir dada por dos aspectos fundamentales, *ver Figura 74:* El primero de ellos es el rango de variación de los

ángulos internos 1 y 2. Estos ángulos serán agudos y como puede observarse en la *Figura 74* los valores que tomen cambian en gran medida el aspecto de la **ZT** de la flecha. El segundo es la longitud de los lados que forman la **ZR**, que harán que el área de esa zona sea mayor o menor.

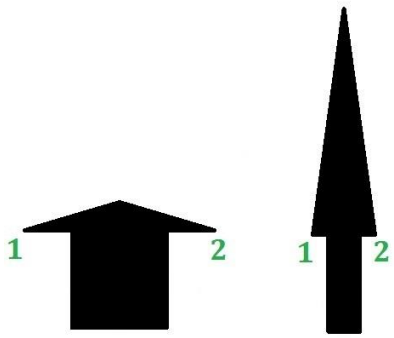


Figura 74. Variación en la forma de flecha

Como se puede ver en la *Figura 75*, el algoritmo consigue detectar flechas dentro de todo el rango de variación de los ángulos 1 y 2. Sin embargo en la flecha de la parte derecha de la imagen, cuyos ángulos 1 y 2 son los mayores se observa que se produce un fallo de reconocimiento. Es decir no se considera que sea una flecha, y por tanto no se calcula su dirección.

Figura 75. Detección y reconocimiento de varias formas de flecha (I)

Esto es debido a que los filtrados que se efectúan en la fase de reconocimiento delimitan la capacidad que tiene el algoritmo solución para captar diferentes formas de flecha. Concretamente uno de los filtros explicados en el *apartado 4.4.3.2.* establece que para que una detección se considere flecha la zona triángulo debe tener entre dos y tres ángulos agudos mientras que la zona rectángulo debería estar formada de entre dos y cuatro ángulos rectos. Considerando ángulos agudos los menores de 80º y ángulos rectos los mayores de 79º y menores de 120º.

La flecha de la parte derecha de la *Figura 75* está formada en su zona triángulo por un ángulo menor que 80º y por dos igual o ligeramente mayores a 80º, por tanto el algoritmo sólo reconoce un ángulo agudo y los otros dos los toma por rectos, es por ello que no lo considera una flecha y descarta la detección.

La razón de seleccionar los intervalos citados para definir los ángulos rectos y obtusos, y no utilizar el valor fijo de que los ángulos rectos miden 90º y los agudos menos que 90º es debido a que la visión por computador no es del todo precisa. Dependiendo del ángulo de la cámara o debido a una segmentación no del todo buena se puede captar un ángulo recto como obtuso o agudo y viceversa. Es por ello que para captar con

fiabilidad el mayor abanico de flechas posibles, se ha tenido que sacrificar la detección de aquellas flechas cuyos ángulos 1 y 2 sean mayores que 80º y menores que 90º.

En conclusión el algoritmo es capaz de detectar todo tipo de fechas con un rango de variabilidad de sus ángulos 1 y 2 mayor que 0º y menor que 80º.

A parte de los tipos de flecha que se han comentado hasta ahora, el algoritmo también es capaz de detectar y reconocer flechas del tipo mostrado en la *Figura 76.*

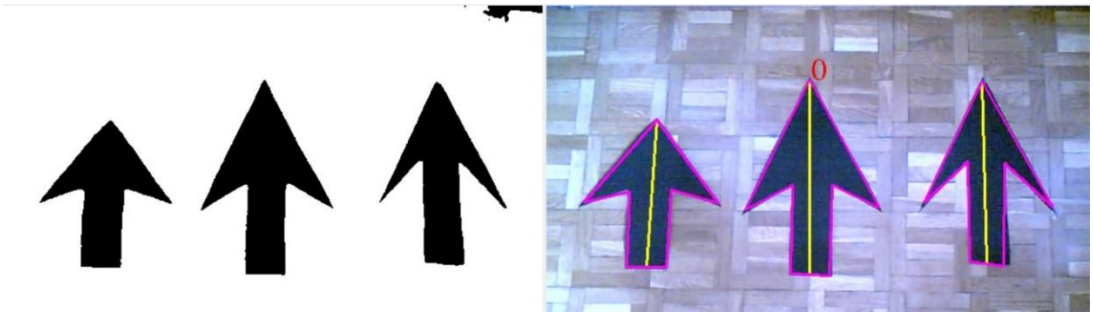


Figura 76. Detección y reconocimiento de varias formas de flecha (II)

O incluso flechas con los bordes redondeados como en el caso de la *Figura 77.* Siendo éstas muy susceptibles a fallos en la detección o a los denominados "parpadeos" de los que se habló en el *apartado 4.4.4.3.*

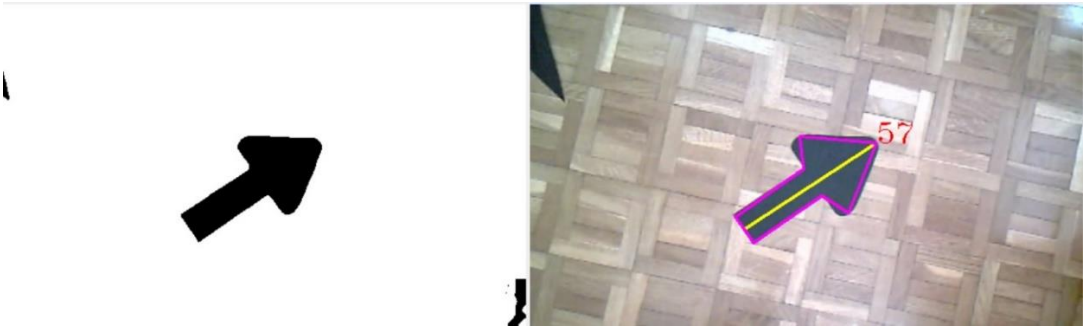


Figura 77. Detección y reconocimiento de varios tipos de flecha (III)

Otro aspecto limitante a la hora de detectar y reconocer flechas aparte del de los ángulos internos, es el del número de lados que conforman la misma. En uno de los filtros del *apartado 4.4.3.2* se establece que sólo se considerarán flechas aquellas formadas por siete vértices, y por lo tanto por siete lados. Es por ello que quedan descartadas de las flechas que el algoritmo es capaz de reconocer aquellas cuyo número de vértices sea superior a siete *ver Figura 78.*

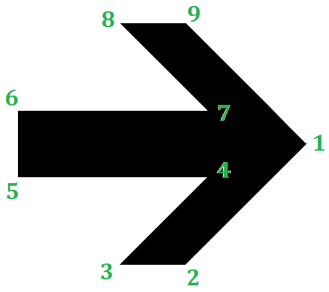


Figura 78. Flecha con número de vértices mayor que siete

Como se vio en el *Capítulo 2* para que los algoritmos ya existentes orientados a la detección y reconocimiento de objetos puedan detectar varios tipos de flecha, necesitarían hacer uso de bases de datos compuestas de plantillas de flechas con diferentes formas. Esto supone limitar en gran medida el abanico de formas de flechas que son capaces de detectar. Además cuantas más formas de flecha se deseen detectar, habrá que incluir un mayor número de plantillas y como se verá en *apartados 5.2.1.3 y 5.2.1.4*, supondrá un detrimento en la velocidad de procesamiento. No ocurre así con el algoritmo propuesto como solución, que al no utilizar bases de datos es capaz de detectar casi ilimitados tipos de flecha (con las excepciones comentadas en este apartado) sin que esto afecte a la velocidad de procesamiento.

5.2.1.4 Tiempo de procesamiento

En este apartado se va a realizar una comparación del tiempo de procesamiento que emplean en la detección y reconocimiento de flechas algoritmos ya existentes como SURF o Template Matching, con el tiempo que emplea el algoritmo que se propone como solución en este apartado. Los tiempos que se van a exponer, por tanto no engloban las demás fases (cálculo de dirección y sentido, cálculo de Yaw etc.)

El procedimiento utilizado a la hora de realizar las pruebas ha sido el siguiente:

Se han tomado datos del tiempo de procesamiento, y se ha hecho simultáneamente una media de los que se disponían hasta el momento. Este valor, el de la media de datos, es el que se va a exponer como resultado de las pruebas en los siguientes apartados.

Cabe comentar también que con el objetivo de que los resultados sean comparables todas las pruebas se han hecho con el mismo ordenador portátil que presenta las siguientes características:

Ordenador portátil Lenovo Flex 2 con procesador Intel Core i3-4010U (1.7GHz, 3MB) dispone de memoria RAM 4GB DDR3L SODIMM (1x4GB) Máx 16GB, el disco duro es de 500 GB. Utiliza sistema operativo Microsoft Windows 8.1 de 64 bits.

La cámara con la que se ha trabajado es la del AR.Drone 2.0. de la marca Parrot, cuyas características principales son: Cámara HD 720p 1280x720 píxeles y 30 FPS *ver apartado 3.2*

- **“Template Matching”:**

Rodeados de azul en la *Figura 79* se muestran los tiempos que ha empleado el método “Template Matching” en las fases de detección y reconocimiento para distintas flechas utilizando una única plantilla. Como se puede observar los tiempos rondan la magnitud de 87 – 91 ms. En la captura de la parte inferior de la figura se muestra que el tiempo empleado cuando aparecen varias flechas en el plano es de 90 ms, y que la correcta detección y reconocimiento sólo se efectúa para la flecha cuya forma es igual a la de la plantilla, siempre y cuando esté en la misma orientación. Para que este método fuera capaz de captar diferentes formas de flecha con diferentes orientaciones debería hacerse uso de bases de datos con numerosas plantillas lo que aumentaría en gran medida el tiempo de ejecución.

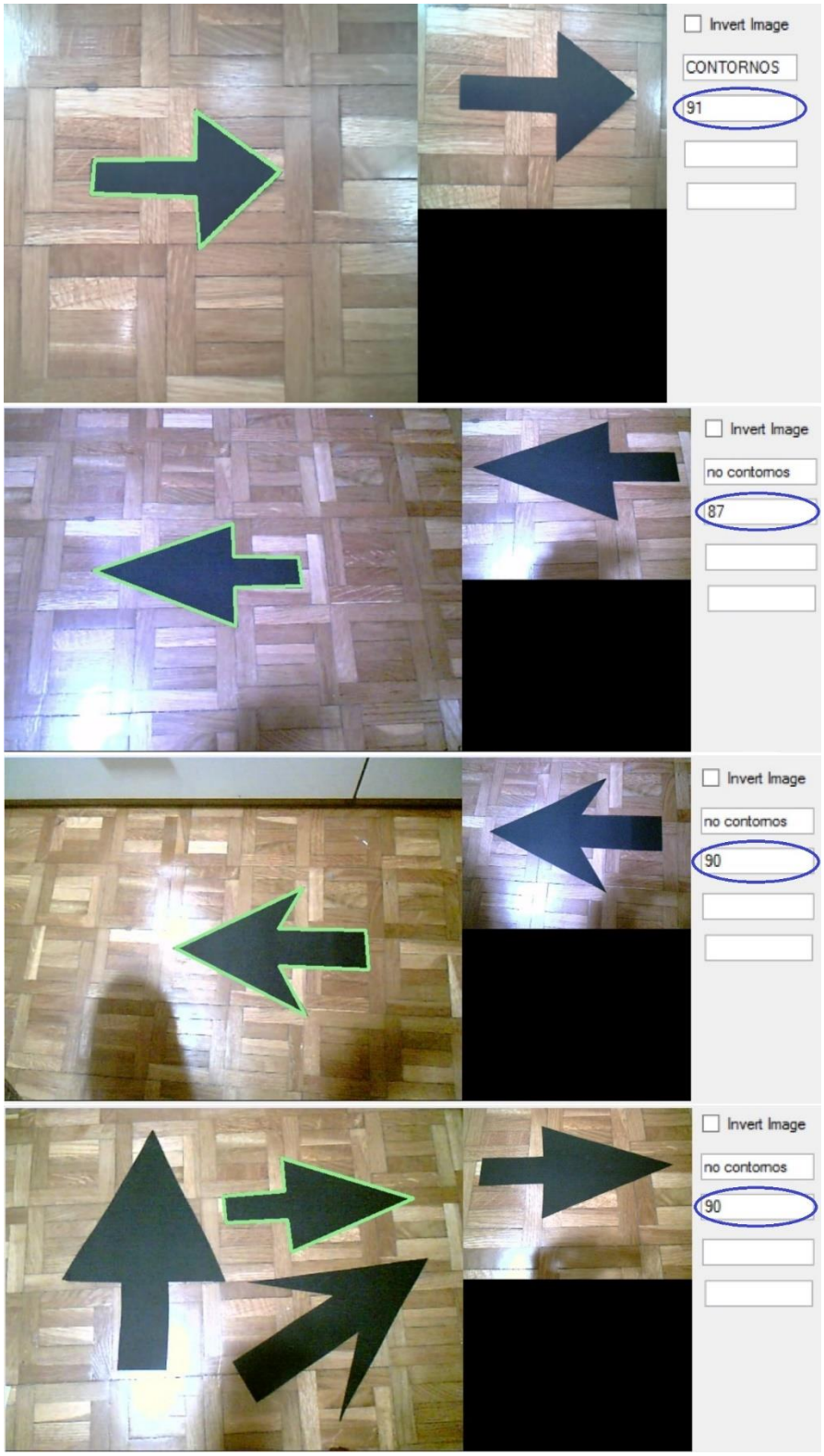


Figura 79. Template Matching: Tiempo empleado en detección y reconocimiento para distintas flechas

- **SURF:**

Rodeados de azul en la *Figura 80* se muestran los tiempos que ha empleado el método SURF en las fases de detección y reconocimiento para distintas flechas

utilizando una única plantilla. Como se puede observar el menor tiempo de ejecución alcanzado corresponde al análisis de una única flecha y es de 194 ms, mientras que el mayor corresponde al análisis de varias flechas en el mismo plano y llega a ser de 318 ms. Además como se puede ver en la figura, SURF presenta serias dificultades para poder detectar objetos simples como en el caso de las flechas, que no presentan ninguna textura. Lo más cerca que ha estado de dar un reconocimiento correcto ha sido en el caso de varias flechas, parte inferior de la *Figura 80.*

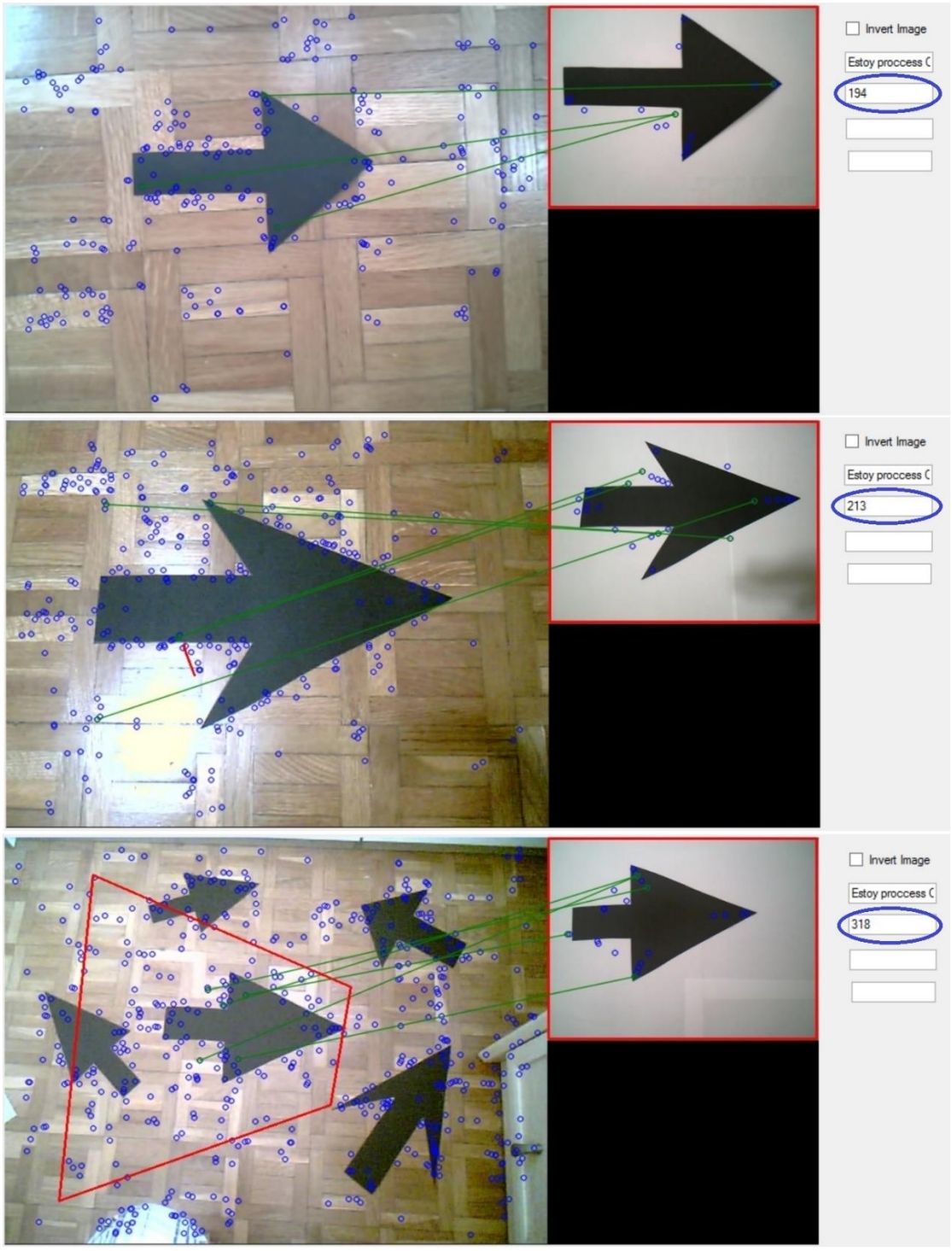


Figura 80. SURF: Tiempo empleado en detección y reconocimiento para distintas flechas

- **Solución propuesta "Markers detection/recognition for UAVS visual navigation systems":**

Como se puede observar en la *Figura 81* rodeado de azul, el tiempo que emplea el algoritmo propuesto como solución en este trabajo es de 8 milisegundos para una única flecha.

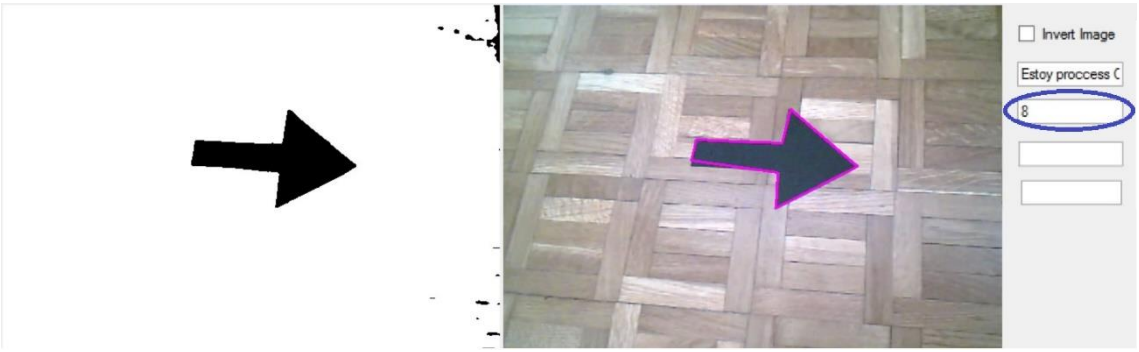


Figura 81. Solución final: Tiempo de procesamiento en las fases de detección y reconocimiento para una flecha

Si se aumenta el número de flechas a detectar *ver Figura 82,* el tiempo empleado pasa a ser de 9 milisegundos.

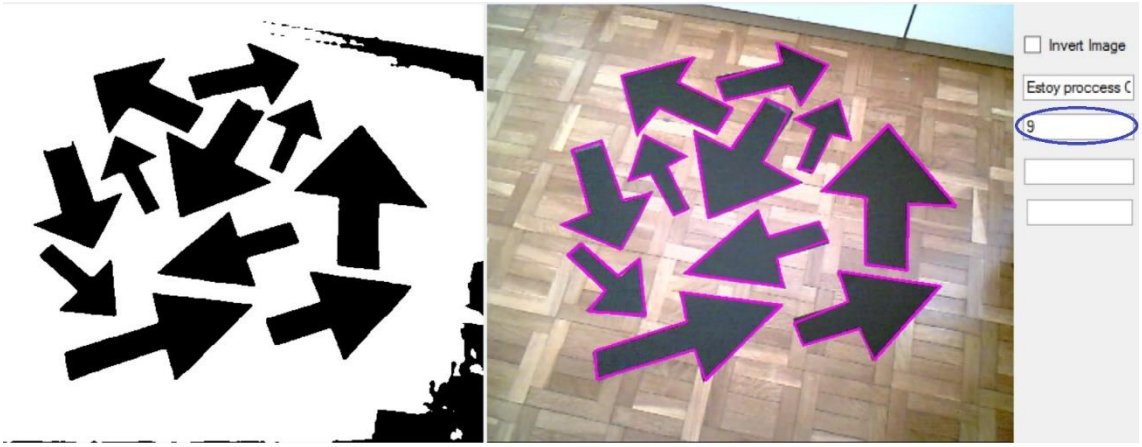


Figura 82. Solución final: Tiempo de procesamiento en las fases de detección y reconocimiento para varias flechas

En este caso al estar analizando sólo la fase de detección y reconocimiento, lo que demuestra que estas fases se han efectuado correctamente es que las flechas aparezcan delineadas por líneas de color morado. Como se puede ver en las *Figuras 81 y 82* con el algoritmo solución se consiguen detectar todas las flechas presentes en el plano.

Comparación:

Los resultados expuestos para los métodos "Template Matching" y SURF, son para el caso de que se esté utilizando una única plantilla. Para que ambos métodos puedan acercarse lo máximo posible a cumplir las características básicas necesarias para llevar a cabo el objetivo de este trabajo necesitarían hacer uso de bases de datos compuestas de numerosas plantillas, lo que supondría un mayor tiempo de

procesamiento que el expuesto en las *Figuras 70* y *80.* Por tanto los resultados expuestos en este apartado muestran los tiempos de procesamiento más bajos que "Template Matching" y SURF pueden alcanzar.

Si al realizar la comparación con los tiempos de procesamiento más bajos que "Template Matching" y SURF son capaces de alcanzar se obtuviera que el algoritmo solución ejecuta en un tiempo menor, se podría asegurar que cuando los métodos existentes utilicen bases de datos para mejorar la detección y reconocimiento, el algoritmo que se propone en este trabajo seguirá siendo más rápido que ellos.

Los resultados expuestos indican que para el caso de detección de una sola flecha, el método "Template Matching" emplea un tiempo de procesamiento más de 11 veces mayor que el propuesto como solución en este trabajo, mientras que el tiempo que emplea SURF llega a ser más de 24 veces mayor. Aun comparando el caso extremo de que el algoritmo propuesto esté realizando el esfuerzo de detectar y reconocer varias flechas y "Template Matching" y "SURF" sólo una con una única plantilla, se observa que el tiempo de procesamiento sigue siendo más de 10 y 21 veces mayor respectivamente que el del algoritmo solución.

En conclusión el tiempo de procesamiento que necesita el algoritmo que se propone como solución en este trabajo es mucho menor que el menor de los tiempos que los métodos "Template Matching" y SURF pueden alcanzar. Incluso si se enfrenta el caso extremo de que el algoritmo solución esté detectando varias flechas mientras que los otros dos sólo detectan una, se ha visto que la diferencia sigue siendo considerable. Por lo tanto se ha conseguido desarrollar un algoritmo mucho más veloz que los que ya existían.

Cabe recordar que el objetivo de este trabajo es superar una carrera de drones autónoma en el que el ganador es aquel dron capaz de recorrer el camino de flechas en el menor tiempo posible. Utilizando el algoritmo propuesto como solución el dron sería capaz de recorrerlo mucho más rápido que utilizando los algoritmos ya existentes, teniendo más probabilidades de resultar vencedor. Esto es debido a que los otros algoritmos son más lentos procesando la información, lo que supone que deba establecerse una velocidad de avance del dron lo suficientemente lenta como para que a estos algoritmos les dé tiempo a efectuar la detección, reconocimiento y análisis de flechas, y a mandar la información al controlador del dron para que efectúe los movimientos correspondientes.

5.2.1.5 Trabajo a tiempo real

Una de las características básicas que deben alcanzarse es que el algoritmo trabaje a tiempo real, para que el dron pueda reaccionar a la información extraída de las flechas al instante y realizar los movimientos correspondientes. Que el dron sea capaz de reaccionar al instante es fundamental, ya que si efectuara los movimientos con retraso podría perder el camino de flechas y no se habría conseguido por tanto cumplir con el objetivo expuesto en el *apartado 1.3.*

Para saber si el algoritmo trabaja a tiempo real se va a comprobar si es capaz de ejecutarse antes de que la cámara adquiera el siguiente frame. Si es así, el algoritmo tendrá tiempo de analizar todos los frames captados por la cámara y el tiempo real estará asegurado. Sin embargo, si el tiempo que tarda en ejecutarse es superior al tiempo de adquisición del siguiente frame, significará que no será capaz de procesar

todos los frames captados por la cámara aumentando el riesgo de que no se trabaje a tiempo real.

El procedimiento de medida utilizado en la obtención de los resultados que se exponen a continuación es el mismo que el explicado en el *apartado 5.2.1.4.* Además debe tenerse en cuenta que en estas pruebas se ha utilizado la cámara del AR.Drone 2.0. de la marca Parrot, cuyas características principales son: Cámara HD 720p 1280x720 píxeles 30 FPS *ver apartado 3.2.*

En las *Figuras 83 y 84* se muestra el tiempo de ejecución total del algoritmo solución para una flecha y para varias respectivamente. Cabe comentar que con varias flechas el esfuerzo de procesamiento es superior y por eso el tiempo de ejecución aumenta un milisegundo.

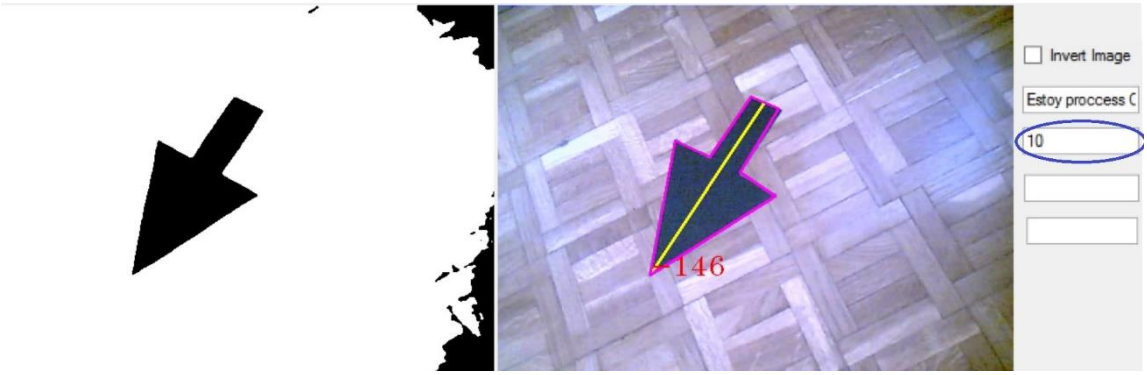


Figura 83. Solución propuesta: Tiempo de procesamiento total para una flecha

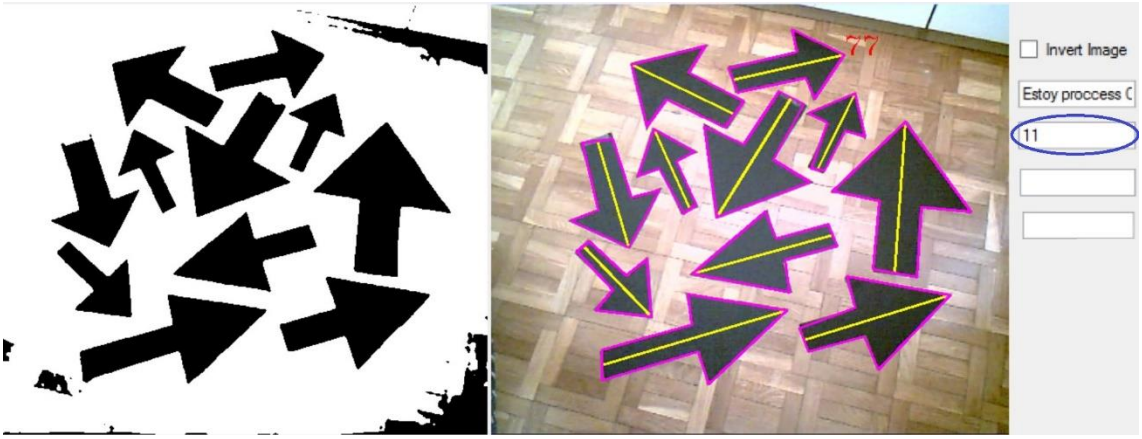


Figura 84. Solución propuesta: Tiempo de procesamiento total para varias flechas

Como la cámara utilizada funciona a 30 FPS, se va a calcular cada cuánto tiempo adquiere un frame *ver Ecuación 5.1:*

$$tiempo\ adquisición\ frame = \frac{1\ frame * 1s}{30\ frames} = 0{,}03333333333\ s = 33{,}33333\ ms \quad \textbf{(5.1)}$$

El resultado de la *Ecuación 5.1* indica que la cámara adquiere un nuevo frame cada 33,33333 ms. Como se ha visto en las *Figuras 83 y 84*, el algoritmo solución tarda 10-11 ms en ejecutarse, por tanto es capaz de procesar todos los frames que adquiere la cámara, ya que termina de ejecutarse completamente mucho antes de que se

adquiera el nuevo frame. Por estas razones se puede afirmar que el algoritmo solución trabaja a tiempo real.

Para los métodos “Template Matching” y SURF se ha visto que el tiempo de ejecución más bajo que alcanzan es 87 ms y 194 ms respectivamente, ver *Figuras 79 y 80.* La cámara adquiere frames cada 33,33333 ms, más rápido de lo que tardan en ejecutarse. Por tanto utilizando estos métodos no pueden procesarse todos los frames que adquiere la cámara, aumentando el riesgo de que no se trabaje a tiempo real. Este riesgo es mayor en el caso de SURF debido a que es notablemente más lento que “Template Matching”

Como se ha indicado en el comienzo de este apartado, las medidas para “Template Matching” y SURF corresponden sólo a la fase de detección y reconocimiento de flechas. Si se les incluyeran el resto de fases correspondientes al análisis de las flechas y extracción de información para el guiado del dron, y se emplearan bases de datos para mejorar la capacidad de detección y reconocimiento de flechas, supondría aumentar aún más el tiempo de procesamiento llegando a ser inviable utilizarlos para superar el problema planteado en este trabajo expuesto en el *apartado 1.2*.

Las conclusiones que se pueden extraer del análisis realizado en este apartado son las siguientes:

- **Algoritmo solución “Markers detection/recognition for UAVS visual navigation systems”**:
  - Trabaja a tiempo real, siendo capaz de procesar todos los frames que capta la cámara.
  - Al no tener que esperar a que el algoritmo termine de realizar el procesamiento del frame cuando ya se ha adquirido el siguiente, las operaciones de captar, procesar la información de las flechas y transmitírsela al dron se producen de manera inmediata. Esto permite recorrer el camino de flechas de forma mucho más precisa reduciendo el riesgo de que el dron se pierda. Además se puede establecer una mayor velocidad de avance lo que supone recorrer el camino de forma rápida, aspecto fundamental en una carrera.
- **Métodos “Template Matching” y “SURF”**:
  - Alto riesgo de no trabajar a tiempo real.
  - Es necesario esperar a que el algoritmo termine de realizar el procesamiento del frame actual una vez que la cámara ya ha adquirido el siguiente, por lo que las operaciones de análisis de flechas y el guiado del dron no se efectúan de manera inmediata. Como consecuencia aumenta el riesgo de que el dron pierda el camino de flechas, y supone un detrimento en su velocidad de avance, porque habría que reducirla de forma tal que al algoritmo le diera tiempo a procesar toda la información y transmitírsela al dron antes de que siga avanzando.

5.2.2 Características extra: Espacio de color

Como se indicó en el *apartado 1.2* las flechas que conforman el recorrido en la competición son todas de un mismo color uniforme y sólido.

Los colores a detectar se establecen con el umbral elegido en la segmentación. Seleccionado los umbrales adecuados se pueden detectar ilimitados colores. Efectuando esta operación de variación de umbral el algoritmo propuesto como solución es capaz de detectar todo tipo de colores, algunos ejemplos de ello se muestran en la *Figura 85*.

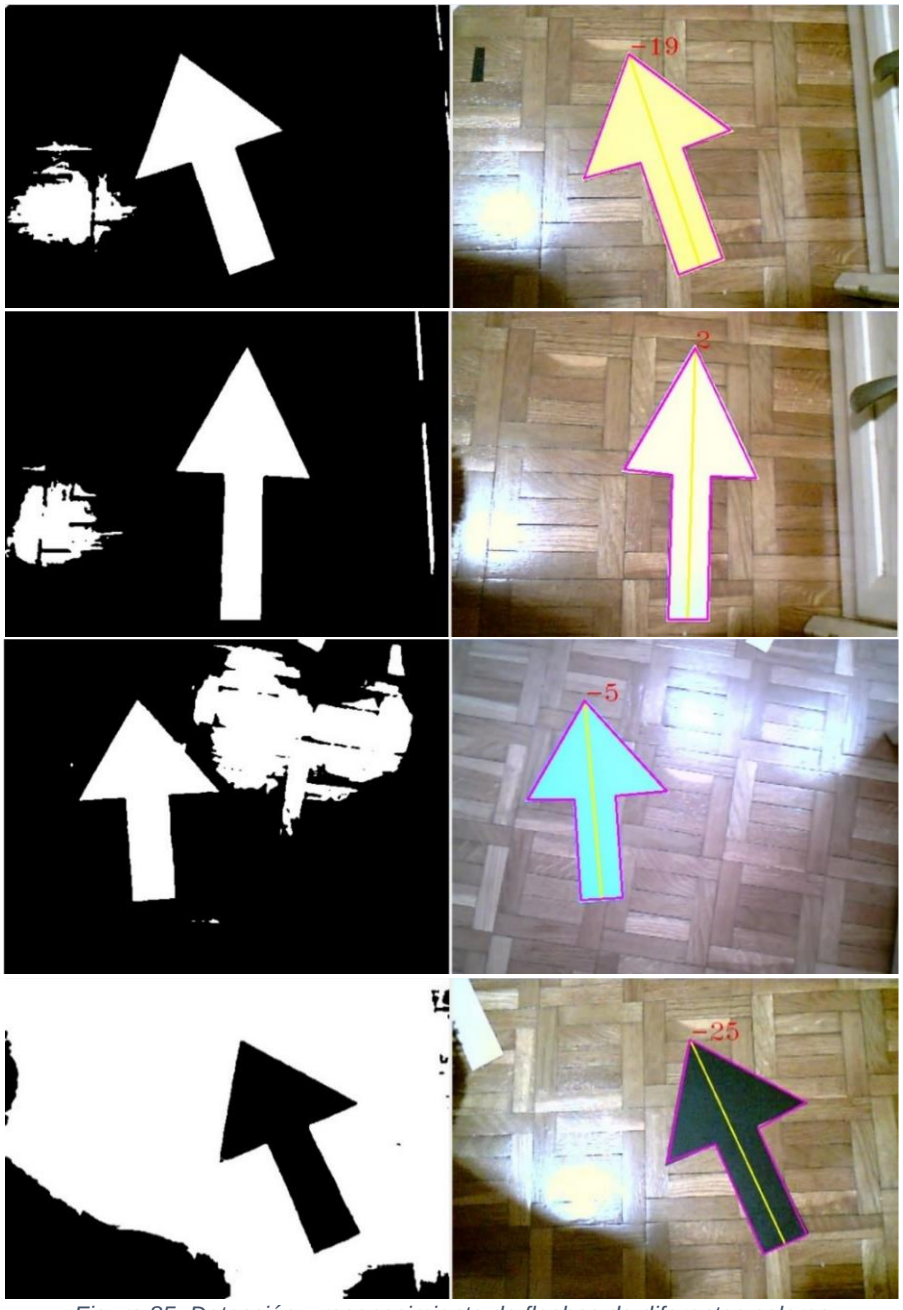


Figura 85. Detección y reconocimiento de flechas de diferentes colores

Cabe destacar que el algoritmo no es capaz de detectar todos los colores a la vez. Si se establece el umbral para detectar flechas de colores blancos por ejemplo, sí será capaz de detectar a la vez otras flechas de colores claros como podría ser rosa pálido o naranja, pero no podrá detectar flechas de colores tan dispares como el negro *ver Figura 86.*

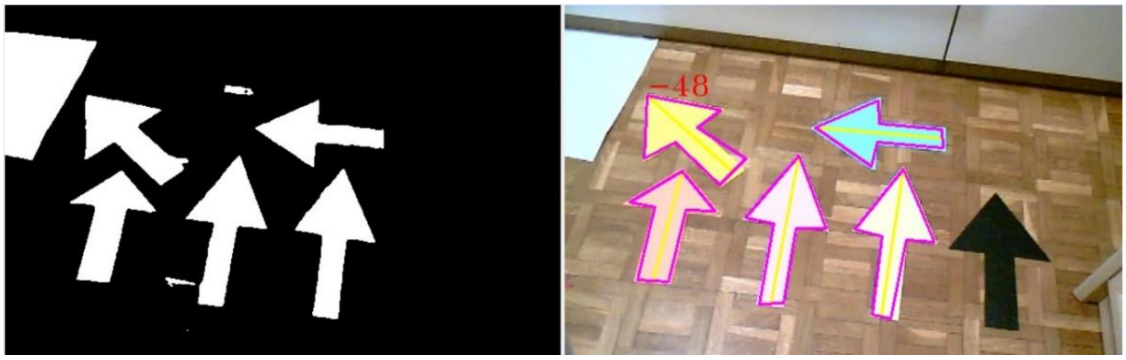


Figura 86. Detección y reconocimiento de flechas de varios colores a la vez

Esto no es un aspecto limitante de cara a la competición, ya que las flechas que conformarán el recorrido serán siempre de un mismo color uniforme y sólido *ver apartado 1.2.* Es por ello que antes de iniciar la competición se podrá elegir el umbral adecuado para que detecte el color de las flechas que conforman el recorrido de manera rápida y sencilla.

5.3 Guiado

En este apartado se van a exponer los resultados obtenidos a la hora de calcular la dirección y ángulo Yaw de las flechas, las pruebas de los métodos implementados para hacer un guiado más robusto, y los resultados de las pruebas finales en las que el dron ha recorrido de forma autónoma varios caminos formados por flechas.

5.3.1 Cálculo de dirección y Yaw

Gracias a los filtrados efectuados en la fase de reconocimiento *ver apartado 4.4.3.2,* se consigue averiguar sin ninguna incertidumbre cuál es el **VD** y cuál el **Vff**, por lo que la dirección de las flechas y por consiguiente el ángulo Yaw siempre consiguen calcularse de forma correcta *ver Figura 87.* En la figura, la línea amarilla es la línea de dirección de la flecha, y el número representado en color rojo es el ángulo Yaw. La razón de que sólo aparezca un ángulo Yaw está indicada en la explicación del *apartado 4.4.4.3.*

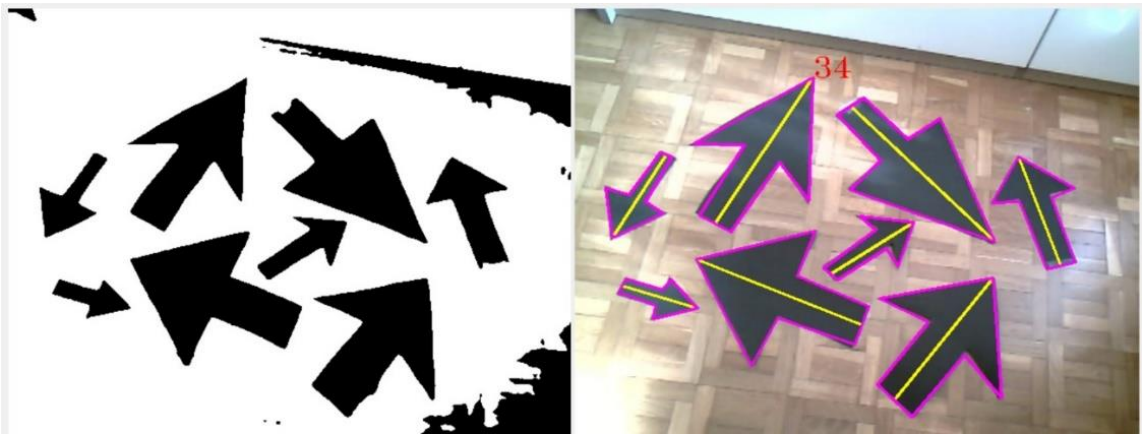


Figura 87. Cálculo de dirección de flecha y ángulo Yaw de la flecha superior del plano

5.3.2 Resultados de los métodos para obtener un guiado más robusto

Como se explicó en el *apartado 4.4.4.3* se han implementado dos métodos para hacer el guiado más robusto, el primero de ellos consiste en guardar el último ángulo Yaw calculado para que en el caso de que se produzca un fallo en el reconocimiento de flecha en un frame, se le envíe al dron un valor de Yaw coherente y parecido al que debería recibir si no se hubiera producido este fallo. En el momento en el que se vuelve a recuperar el reconocimiento de la flecha se le enviará el valor actual. Con esto se consigue que el giro del dron sea coherente y continuo. En la *Figura 88* se puede observar que se ha conseguido implementar con éxito éste método.

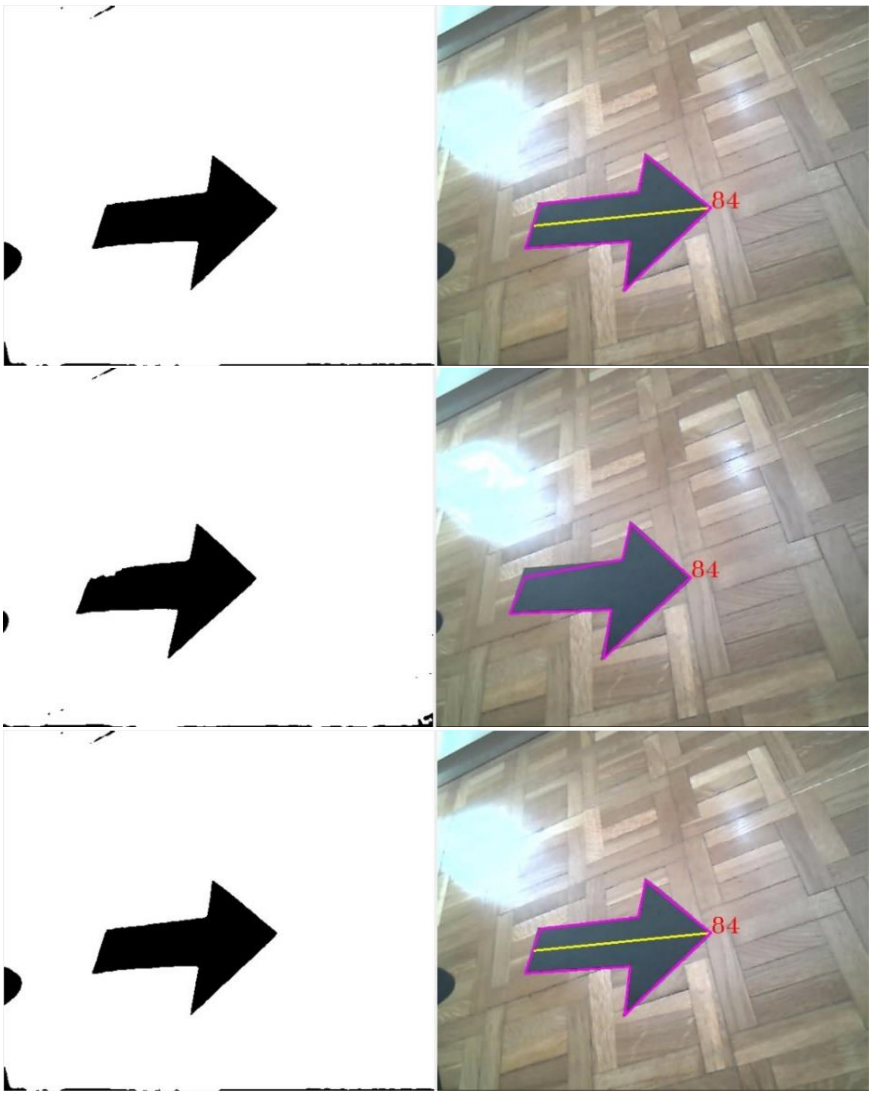


Figura 88. Fallo en el reconocimiento de flecha en un frame

El segundo de los métodos calculados consiste en calcular únicamente el ángulo Yaw de la última flecha detectada. Como el dron avanza hacia delante siempre será la que se encuentre en la parte superior del plano que capta la cámara. La finalidad de esto es no transmitir al dron ángulos de flechas que ya se han analizado. En la *Figura 89* puede observarse que se ha implementado con éxito éste método.

Figura 89. Cálculo de Yaw de la última flecha detectada

Además en este último método se efectúa algo parecido al método explicado al principio de este apartado. Si se produjera un fallo en la detección y reconocimiento de la última flecha detectada, el algoritmo consideraría que la detección más reciente es la de la flecha inmediatamente inferior, ya que es la que se encuentra más arriba en el plano de la cámara en ese momento. Por tanto sería su ángulo Yaw el que se enviara al dron provocando un fallo en el guiado del mismo. Para solucionarlo lo que se hace es guardar el último ángulo Yaw calculado de la flecha más reciente. En caso de fallo sería ese valor el que se enviara al dron, hasta que se volviera a recuperar la detección y reconocimiento de la flecha, momento en el cuál se mandaría otra vez el valor actual. Como se puede observar en la *Figura 90* se ha conseguido implementar con éxito éste aspecto.

Figura 90. Fallo en la detección de la última flecha detectada

5.3.3 Resultados

En este apartado se van a exponer los resultados obtenidos en las pruebas finales donde se simuló la competición para la que se ha desarrollado el algoritmo propuesto como solución.

Se hicieron un total de 20 pruebas en las que se construyeron varios recorridos formados por flechas de diferentes tipos, y con un número variable de flechas, algunos estaban formados por ocho flechas, otros por cinco etc. En total se procesaron 167 flechas.

Los resultados que se van a exponer van a estar divididos en:

- **Falsos positivos**: Reconocer como una flecha algo que no lo es.
- **Fallos en detección**: No detectar flecha por una mala segmentación.
- **Fallos en reconocimiento**: La flecha no supera los filtros del reconocimiento.
- **Fallos en guiado**: Engloba fallos en los métodos implementados para hacer el guiado más robusto, o fallos en el control del dron.
- **Ejecución correcta**: Correcto guiado del dron.

Tabla 5. Resultados

Falsos positivos	2	1,20%
Fallos en detección	1	0,60%
Fallos en reconocimiento	4	2,40%
Fallos en guiado	7	4,19%
Ejecución correcta	153	91,62%
Total	167	100%

Figura 91. Resultados finales

Como se observa en la *Tabla 5* y en la *Figura 91* con el algoritmo desarrollado se obtiene un alto porcentaje de ejecuciones correctas. Dentro de los fallos de menor a mayor porcentaje se encuentran:

- **Fallos en la detección**: El porcentaje es bajo debido a que se realizó una buena segmentación. Tan sólo se produjo un fallo debido a un reflejo de luz que se encontraba muy próximo a la flecha.
- **Falsos positivos**: Debido a sombras del suelo con forma parecida a las de las flechas que consiguen pasar los filtros de la fase de reconocimiento. En este caso el número de fallos también es reducido porque gracias a la segmentación realizada se consigue eliminar gran parte del fondo y de éstas sombras.

- **Fallos en reconocimiento:** Se produjeron en las flechas cuya forma es más susceptible a no ser reconocida, concretamente en flechas con los bordes redondeados y aquellas cuyos dos ángulos de la zona triángulo son mayores a 80º ver *apartado 5.2.1.3*
- **Fallos en guiado:** El mayor porcentaje de fallo está concentrado en el guiado, ya que dentro de él se contempla el control del dron. Los errores que se han dado han sido debidos a que el dron no ha conseguido frenar en el intervalo establecido, y ha terminado ligeramente desviado de la dirección que marcaba la flecha. Las razones de esto pueden ser dos, la primera es que como se vio en el *apartado 4.4.4.4* hay veces que es necesario hacer uso de la brújula que el dron lleva incorporada, que al no ser del todo precisa provoca estas desviaciones. La segunda es que aunque los intervalos de frenado se han elegido de acuerdo a las velocidades de avance y giro, no siempre son infalibles y también puede provocar que se frene ligeramente más tarde.

Cabe destacar que cuando se producen estos fallos hay dos posibilidades, que el dron consiga detectar la siguiente flecha y pueda corregir su dirección, por lo que será capaz de recuperar el camino de flechas sin ningún problema o habiendo perdido algo de tiempo, o que alguno de estos fallos haya supuesto que el dron termine el giro en una dirección donde no ha encontrado más flechas y se haya perdido, por lo que en el caso de la competición real no hubiera conseguido acabar la carrera.

Cuanto más juntas se coloquen las flechas que forman el recorrido más probabilidades hay de que aunque se produzca un fallo en una de ellas el dron pueda encontrar la siguiente y no perderse. En las pruebas realizadas, en tres ocasiones estos fallos supusieron que el dron no consiguiera terminar el recorrido de flechas, en cinco de ellos se produjeron uno o varios fallos pero el dron consiguió encontrar la siguiente flecha corregir su dirección y terminar la carrera, y en los doce restantes la ejecución se produjo correctamente.

Capítulo 6: Presupuesto

6.1 Introducción

En este capítulo se detallan los costes que ha supuesto el desarrollo de este proyecto.

El presupuesto total se va a dividir en dos capítulos, coste de materiales y salario del desarrollador. Dentro de los materiales se contemplan el ordenador utilizado para el desarrollo del algoritmo, el drone usado en las pruebas del algoritmo, y un conjunto denominado "varios" dentro del cual se incluyen los gastos en las cartulinas que se utilizaron para hacer las diferentes flechas que se utilizaron en las pruebas, y la cinta adhesiva que se utilizó para fijarlas al suelo.

6.2 Coste de materiales

Debido a que el producto final es el algoritmo, no se va a incluir el precio total de los equipos que se han utilizado para su desarrollo (ordenador y drone) ya que no son adquisiciones nuevas ni se van a vender al cliente, en su lugar se va a incluir el coste de amortización de los mismos. No ocurre así con el grupo "varios" que sí es una adquisición nueva y necesaria para realizar las pruebas, por lo que su importe total sí se considera gastos del proyecto

Para calcular el coste de amortización se va a tener en cuenta el precio real al que se adquirieron los equipos, el coeficiente indicado en las últimas tablas de amortizaciones publicadas por el Ministerio de Hacienda [4], el porcentaje de uso que se les ha dado a los equipos y la duración total del proyecto que como se indicó en el *apartado 1.5* es de ocho meses. *Ver Ecuación 6.1.*

$$Coste\ desamortización = Precio * coef.amortización.* porcetaje\ uso * \frac{8\ meses}{12\ meses} \quad \textbf{(6.1)}$$

En el caso de este proyecto, se ha utilizado más tiempo el ordenador que el dron, pero en una proporción no muy dispar por lo que se establecen unos porcentajes de uso 55% - 45% *ver Tabla 6:*

Tabla 6. Tabla resumen de precio total, coeficientes de amortización y uso, y coste de desamortización

Equipos	Precio total (€)	coef. Amortización (%)	Porcentaje uso (%)	Coste amortización(€)
Ordenador	526,99	20	55	38,65
Dron	249	15	45	11,20

El total del coste de material sería la suma del coste de amortización y el coste total del grupo "varios" *ver Tabla 7:*

Tabla 7. Tabla resumen del coste de material

Materiales	Coste materiales (€)
Ordenador	38,65
Dron	11,20
Varios	5,50

6.3 Salario del desarrollador

Se establece un salario para el desarrollador de 12 euros brutos por hora trabajada.

Como se puede observar en el *apartado 1.5* la duración de este proyecto ha sido de ocho meses (Diciembre fue no laboral). Las horas empleadas en cada mes se distribuyen de la siguiente manera:

De Octubre a Marzo se emplearon $2h/día$ sin incluir fines de semana. Teniendo en cuenta que en diciembre no se trabajó, el cálculo del salario queda de la siguiente manera *ver Ecuación 6.2*:

$$salario_{Octubre-Marzo} = 12\frac{€}{h} * 2\frac{h}{día} * 20\frac{días}{mes} * 5\ meses = 2400€ \quad \textbf{(6.2)}$$

De Abril a Junio se emplearon $5h/día$ incluyendo fines de semana. El cálculo es el siguiente *ver Ecuación 6.3 y 6.4*:

$$salario_{Abril\ y\ Junio} = 12\frac{€}{h} * 4\frac{h}{día} * 30\frac{días}{mes} * 2\ meses = 2880€ \quad \textbf{(6.3)}$$

$$salario_{Mayo} = 12\frac{€}{h} * 4\frac{h}{día} * 31\frac{días}{mes} = 1488€ \quad \textbf{(6.4)}$$

Por tanto el coste total derivado del salario del desarrollador durante todo el proyecto ha sido *ver Ecuación 6.5*:

$$coste = salario_{Octubre-Marzo} + salario_{Abril\ y\ Junio} + salario_{Mayo} = 6768\ € \quad \textbf{(6.5)}$$

6.4 Presupuesto total

En este apartado se incluye la tabla resumen que recoge el presupuesto total dividida por capítulos:

CÓDIGO	UNIDAD	DESCRIPCIÓN	CANTIDAD	PRECIO (UNITARIO)	IMPORTE
	CAPÍTULO 1. MATERIALES				
01-001	uds	ORDENADOR MARCA LENOVO			
		Ordenador portátil Lenovo Flex 2 con procesador Intel Core i3-4010U (1.7GHz, 3MB) dispone de memoria RAM 4GB DDR3L SODIMM (1x4GB) Máx 16GB, el disco duro es de 500 GB, su pantalla es táctil de 14" Full HD (1920 x 1080) 16:9. Utiliza sistema operativo Microsoft Windows 8.1 de 64 bits con dimensiones 343mm x 255 mm	1	526,99 €	38,65 €

01-002	uds	DRONE			
		AR Drone 2.0 de la marca Parrot de dimensiones 77,7 x 38,3 x 12,5 mm y peso 31 g. Está equipado con sensor de presión, giroscopio, acelerómetro y magnetómetro de tres ejes, cámara de vídeo HD 720p y 30 FPS. Incluye batería recargable de Li-Po de tres elementos 1000 mAh	1	249 €	11,20 €
01-003	uds	VARIOS			
		Incluye los materiales utilizados en las pruebas del algoritmo (cartulinas, tijeras y cinta adhesiva)	1	5,50 €	5,50 €
TOTAL CAPÍTULO 1.					55,35 €
	CAPÍTULO 2. PERSONAL				
02-001	meses	Incluye el salario del ingeniero a cargo de desarrollar el proyecto. Se le asigna un sueldo de 12 €/h, la duración total del proyecto ha sido de ocho meses.	8	6.768 €	6.768 €
TOTAL CAPÍTULO 2.					6.768 €
TOTAL					6823, 35 €

Capítulo 7: Conclusiones y líneas futuras

7.1 Conclusiones

Se ha conseguido desarrollar un algoritmo para competiciones de drones autónomos que es capaz de detectar, reconocer, analizar y extraer información de flechas, y de transmitir ésta información al dron con el fin de que pueda seguir un recorrido formado por las mismas de manera autónoma. El algoritmo desarrollado también incluye el control que permite que los movimientos que el dron efectúa se hagan de forma correcta y lo más precisa posible.

Se ha conseguido que el algoritmo sea invariante a la rotación y a la escala, sea capaz de detectar casi ilimitados tipos de flecha, trabaje a tiempo real, y que además lo haga empleando poco tiempo de procesamiento, 11 ms. La alta velocidad de procesamiento unido a la alta tasa de ejecuciones correctas, 91.62%, hace que el algoritmo propuesto como solución sea robusto, fiable y con altas posibilidades de ser el que recorra el camino de flechas en el menor tiempo, resultando por tanto vencedor en la carrera.

Conseguir un tiempo de procesamiento tan reducido, ha sido posible gracias a que el método de detección y reconocimiento utilizado está basado en la forma, y no se ha hecho uso de bases de datos que ralentizan en gran medida el tiempo de ejecución. Es por ello que algunos de los algoritmos ya existentes de detección de objetos como pueden ser “SURF”, “SIFT” o clasificadores como el tipo Haar no consiguen alcanzar el rendimiento y la calidad de ejecución del algoritmo propuesto como solución. Estos algoritmos existentes basan su método de detección y reconocimiento en bases de datos que además de limitar el número de tipos de flechas que pueden detectar, supone un coste computacional que ralentiza en gran medida el tiempo empleado en el procesamiento. En estos algoritmos, cuanto mayor es el número de tipos de flecha contemplados mayor es el detrimento en la velocidad de procesamiento. Además, algunos de ellos llegan incluso a presentar dificultades a la hora de detectar objetos en diferentes orientaciones y escalas.

El coste total que ha supuesto el desarrollo de este proyecto ha sido de 6.823,35 €, siendo el 99,2 % de ese coste correspondiente al salario del ingeniero durante los meses de desarrollo del proyecto. Es por ello que el coste real que supondría implementar el algoritmo en un dron sería muy bajo, sólo se necesitaría adquirir un dron con cámara en el caso de que no se tuviera.

Por último, es importante destacar que el algoritmo desarrollado puede implementarse en cualquier dron, con la única restricción de que cumpla las siguientes características básicas: que esté equipado con cámara orientada u orientable hacia el suelo para poder percibir las flechas, que lleve incorporado un magnetómetro, y que debido a que

la competición se puede desarrollar tanto en exterior como en interior, el dron elegido deberá tener un tamaño y rango de velocidades adecuado para que se pueda volar en interiores.

Por tanto por un coste reducido se podría disponer de un algoritmo para una competición de drones autónomos que reportaría al dron que lo implemente altas probabilidades de resultar vencedor.

7.2 Futuros trabajos

A continuación se van a exponer las dos principales líneas de trabajo que se podrían seguir para mejorar el algoritmo propuesto como solución.

La primera de ellas tiene que ver con los tipos de flechas que se pueden detectar. Como se ha visto en los *apartados 4.4.3.2* y *5.2.1.3* al ser un algoritmo basado en la forma, los filtros establecidos en la fase de reconocimiento son los que establecen qué detecciones se consideran flecha y cuáles no. Aunque con los filtros que se han establecido en el algoritmo actual se consigue abarcar un número muy grande de tipos de flecha, hay algunas como las compuestas por un número mayor a siete vértices que no es capaz de reconocer como flecha *ver Figura 92*. Es por ello que una de las líneas futuras de trabajo que se podrían alcanzar sería la de refinar los filtros de la fase de reconocimiento para poder abarcar los tipos de flecha que el algoritmo actual no es capaz de reconocer. Para ello una solución podría ser implementar lo siguiente: si las flechas están compuestas por siete vértices se aplicarían los filtros que existen ahora. Si por el contrario están formadas por un número diferente de vértices, se podría aplicar la misma filosofía de los filtros existentes (número y tipo de ángulos internos de cada zona, paralelismo etc.) pero adecuando los valores a las características de la flecha formada por ese número de vértices. En el ejemplo de la *Figura 92* el filtro de ángulos debería contemplar que la flecha está compuesta por un total de nueve ángulos internos, que en la **ZT** debe haber tres ángulos agudos y dos obtusos etc.

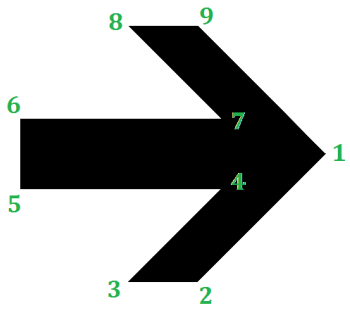


Figura 92. Flecha que con los filtros actuales no es reconocida

La segunda línea de trabajo futura tiene que ver con los colores de flecha que el algoritmo es capaz de detectar. Las flechas que conforman el recorrido en la competición son todas de un mismo color uniforme y sólido *ver apartado 1.2*. Es por ello que el método seguido actualmente consiste en ajustar el umbral de la segmentación antes de la carrera para que detecte el color de las flechas que forman el camino. Ésta operación es sencilla, rápida y efectiva debido a que todas las flechas del recorrido son del mismo color. Sin embargo si en la prueba se mezclaran flechas de diferentes colores, con el método actual el algoritmo sólo sería capaz de detectar aquellas que tuvieran tonalidad parecida al del color para el que se ha establecido el

umbral *ver apartado 5.2.2.* Con el fin de hacer el algoritmo robusto ante una posible prueba en la que las flechas del recorrido fueran de colores diferentes, se propone estudiar un método que permita detectar diferentes colores a la vez. Una manera de conseguirlo sería en lugar de utilizar el método actual de variación del umbral de segmentación, jugar con las características de los diferentes espacios de color RGB, HSV etc. y ver con cuál de ellos se consigue alcanzar esta meta.

Capítulo 8: Bibliografía

[1] «Amazon Web Services» [En línea]. Recuperado de: http://s3.amazonaws.com/media.wbur.org/wordpress/10/files/2015/08/0808_oag_drones-starting.jpg. [Último acceso: 17 05 2016].

[2] BAY, Herbert; TUYTELAARS, Tinne; VAN GOOL, Luc. Surf: Speeded up robust features. En *Computer vision–ECCV 2006*. Springer Berlin Heidelberg, 2006. p. 404-417.

[3] BOE (2014). Real Decreto-ley 8/2014, de 4 de julio, de aprobación de medidas urgentes para el crecimiento, la competitividad y la eficiencia. BOE 163 (5 Julio 2014): 52544 – 52715.

[4] BOE (2004). Real Decreto 1777/2004, de 30 de julio, por el que se aprueba el Reglamento del Impuesto sobre Sociedades. BOE 189 (18 05 2016): BOE-A-2004-14600. Modificada el 11 de Julio de 2015.

[5] «BoofCV» [En línea]. Recuperado de: http://boofcv.org/images/thumb/9/99/Object_contours.jpg/631px-Object_contours.jpg. [Último acceso: 16 05 2016]

[6] DALAMAGKIDIS, Konstantinos; VALAVANIS, Kimon P.; PIEGL, Les A. *On integrating unmanned aircraft systems into the national airspace system: issues, challenges, operational restrictions, certification, and recommendations*. Springer Science & Business Media, 2011.

[7] DOUGLAS, David H.; PEUCKER, Thomas K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature.*Cartographica: The International Journal for Geographic Information and Geovisualization*, 1973, vol. 10, no 2, p. 112-122.

[8] «foto321» [En línea]. Recuperado de: http://foto321.com/3720-thickbox_default/hexacoptero-dji-s900-drone.jpg. [Último acceso: 16 05 2016]

[9] HARTLEY, Richard; ZISSERMAN, Andrew. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[10] HOLDEN, Windsor; MCQUEEN, Dave. *Drones: Consumer & Commercial Applications*, *Regulations & Opportunities 2015-2020*. Juniper, 2015.

[11] HORTON, Michael. *Multiple prediction combination and confidence measures for marine object detection*. 2009. Tesis Doctoral. University of Tasmania.

[12] «Inteligencia Dynamics» [En línea]. Recuperado de: http://www.iuavs.com/pages/aplicaciones_y_usos. [Último acceso: 16 05 2016]

[13] Ley 48/1960, de 21 de julio, sobre Navegación Aérea. BOE 176 (23 Julio 1960): BOE-A-1960-10905. Modificada el 17 de Octubre de 2014.

[14] «Mail online» [En línea]. Recuperado de: http://i.dailymail.co.uk/i/pix/2015/06/16/01/29A6AAB100000578-0-Mark_Cocquio_who_has_been_flying_FPV_First_Person_View_quadcopte-a-35_1434413446992.jpg. [Último acceso: 16 05 2016].

[15] «OpenCV» [En línea]. Recuperado de: http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html. [Último acceso: 16 05 2016].

[16] RAMER, Urs. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1972, vol. 1, no 3, p. 244-256.

[17] RUSINOL, Marçal; DOSCH, Philippe; LLADÓS, Josep. "Boundary Shape Recognition Using Accumulated Length and Angle Information". Joan Martí, José Miguel Benedí, Ana Maria Mendonça y Joan Serrat. Third Iberian Conference on *Pattern Recognition and Image Analysis* IbPRIA 2007, Jun 2007, Girona, Spain. Springer-Verlag, 4478 (4478), pp. 210-217, 2007, Lecture Notes in Computer Science; Pattern Recognition and Image Analysis.

[18] SHARP, John. *Microsoft visual C# 2008 step by step*. Pearson Education, 2007.

[19] SHI, Shin. *Emgu CV Essentials*. Packt Publishing Ltd, 2013.

[20] «Sigis» [En línea]. Recuperado de: http://www.sigis.com.ve/images/productos/uav_x8.jpg. [Último acceso: 16 05 2016].

[21] MONTERO, Andres Solis; STOJMENOVIĆ, Milos; NAYAK, Amiya. Robust detection of corners and corner-line links in images. En *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. IEEE, 2010. p. 495-502.

[22] «Tablet Zona» [En línea]. Recuperado de: http://tabletzona.es/app/uploads/2013/02/AR-Drone-20-Parrot.jpg. [Último acceso: 16 05 2016].

[23] VIOLA, Paul; JONES, Michael. Rapid object detection using a boosted cascade of simple features. En *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. IEEE, 2001. p. I-511-I-518 vol. 1.

[24] VIOLA, Paul; JONES, Michael. Robust real-time object detection. *International Journal of Computer Vision*, 2001, vol. 4.

[25] WINDJU, Amy; MITCHEL, Carl. and WHEELOCK, Clint. *Drones for Commercial Applications*. Tractica, 2015.

[26] «Xataka» [En línea]. Recuperado de: http://www.xataka.com/especiales/siete-drones-de-competicion-listos-para-sacarlos-a-hacer-carreras. [Último acceso: 16 05 2016].